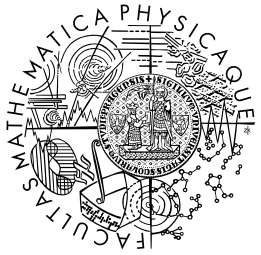


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Tomáš Krejčí

Lifelong localization of robots

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	prof. RNDr. Roman Barták, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I am dedicating this thesis to my dearest girlfriend Natalia. You were here to help me the whole time, providing invaluable help and countless consultations. Without you, this work would be half as good as it is.

I would like to thank my supervisor prof. RNDr. Roman Barták, Ph.D. for his help and time spent guiding this thesis.

Last but not least, I would like to thank Ing. Libor Přeučil, CSc. and Ing. Jan Chudoba, both associated with Intelligent and Mobile Robotics Group, Czech Technical University in Prague, for allowing me to access the robotics hardware and helping with experiments.

Title: Lifelong localization of robots

Author: Tomáš Krejčí

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This work presents a novel technique for lifelong localization of robots. It performs a tight fusion of GPS and Multi-State Constraint Kalman Filter, a visual-inertial odometry method for robot localization. It is shown in experiments that the proposed algorithm achieves better position accuracy than either GPS and Multi-State Constraint Kalman Filter alone. Additionally, the experiments demonstrate that the algorithm is able to reliably operate when the GPS signal is highly corrupted by noise or even in presence of substantial GPS outages.

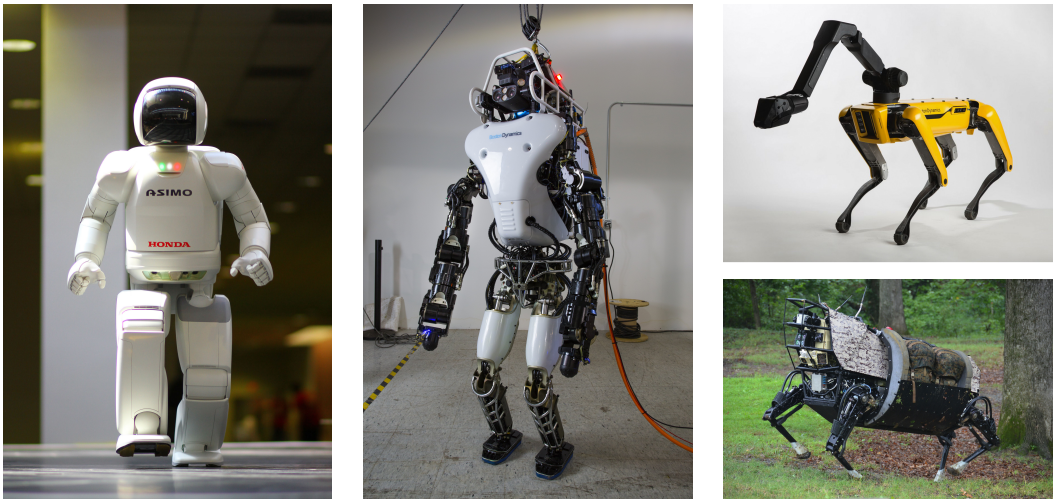
Keywords: localization, odometry, robotics

Contents

Introduction

> *"Robotics is the science of perceiving and manipulating the physical world through computer-controlled mechanical devices."*
>
> - Probabilistic Robotics [1]

When talking about robots, one might think of big and advanced research robots like the ones in figure 1. For many purposes, including this work, any device equipped with processor and sensors allowing it to percept its surroundings is considered a robot, including a smartphone.



(a) Honda Asimo (b) Boston Dynamics Atlas (c) Boston Dynamics SpotMini

Figure 1: A few examples of advanced research robots.

The topic of this work is lifelong localization of robots. The purpose of robot localization is to answer the question *Where am I?*

The answer depends on the purpose of the question. In some cases, a rough estimate like *In Prague* is sufficient. In other cases a more precise answer such as *Exactly at the coordinates x, y, z* is needed. An important part of the answer is also *relative to what* the location is given (its frame of reference). For example a pipe-inspection robot will likely need a very precise estimate of the position in the pipe but the exact location of the pipe in the world might be irrelevant for it. On the other hand, a satellite navigation in a car (also considered a robot) can work with less accurate position but it must be relative to a whole world, not just its surroundings.

The *lifelong* part of the topic refers to the ability to provide position estimate at any time, independently of how long the robot has been operating. Lifelong might mean anything from a few seconds up to the *always-on* mode when the robot operates continuously and localization needs to be available during the whole operation.

This area has been studied for at least 40 years. During this period, a lot of research has been done, covering both theoretical and practical results.

Applications

Robot localization is a problem of many applications. They range from everyday things to experimental projects.

An example everyday application is *augmented reality*, whichgained lots of traction lately. The increased interest was mostly caused by Apple's ARKit [2] and Google's ARCore [3]. In augmented reality, the view of the camera is partially augmented with computer-generated graphics. The core of augmented reality is an accurate motion tracking system. It drives the virtual camera used to render the virtual scene which is then overlaid on top of the camera image.

On the other end of the spectrum might be the *self-driving cars* [4, 5, 6] which use the localization to keep the car in the correct lane, help it to perceive the surroundings, predict the movement of nearby cars and pedestrians and possibly other tasks.

The list of applications also contains precision agriculture [7], drone-based delivery [8], in-warehouse equipment tracking [9], indoor navigation on the airports, hospitals or other public buildings [10], find-and-rescue operations [11], proximity marketing [12] and many other areas.

Goals

This work will focus on the problem of lifelong robot localization, without any assumptions on the operational area of the robot.

It introduces a novel algorithm for a tight fusion of the Global Positioning System (GPS) measurements and visual-inertial odometry (using a single camera and the inertial measurement unit (IMU)). The proposed technique is then thoroughly validated on real-world experiments.

1. Related works

This chapter presents some relevant algorithms from the area of robot localization. To better understand the differences among them, different criteria for their categorization are discussed first.

1.1 Categorization

Before the particular algorithms are discussed, it is helpful to understand different kinds of algorithms, how they differ and what are the consequences of choosing an algorithm from each category.

Note that many methods do not fall into one bucket for each category. They are commonly described in a way that allows them to fall into multiple different buckets, and the particular choices may vary for different implementations.

1.1.1 SLAM and odometry

Two important categories of the algorithms are Simultaneous Localization and Mapping (SLAM) based methods and odometries.

In SLAM based methods, a map of landmarks (usually visually distinct points in the image) is being built and maintained, with the robot localizing in it at the same time. The map size grows, as the robot is exploring new areas. It can be very consistent in some areas and less in others. Especially when the robot explores new space, the position estimate drifts (as there is no map yet) and the map quality degrades. Revisiting the same place again imposes new constraints on the map, making it more accurate. The major drawback is that SLAM complexity grows with map size. For some methods, both time and memory complexity grow with the map size. Others grow only in memory, whereas time complexity remains constant. In cases where the robot does not revisit the same places many times (i.e. a car traveling from one place to another), the added value of having a map diminishes.

On the other hand, odometries typically use only constant size information, without building a map. It might be information from the last few seconds or from the last few camera frames. This allows for straightforward lifelong operation. Probably the biggest drawback is that without a sensor like GPS, the estimation error of current position grows in time indefinitely even when the robot revisits the same area again.

1.1.2 Filtering and iterative minimization techniques

Filtering techniques are based on the recursive Bayesian estimation. They work by estimating a next state of the system from the previous state only. Each state is estimated only once, after which it does not change. On the other hand, iterative minimization techniques are optimizing an error function involving the whole history or some part of it. They can provide a new estimate for past states at every iteration.

Historically, the majority of localization methods fell into the category of filtering-based techniques. They utilize recursive probabilistic estimators like Kalman filter, extended Kalman filter, unscented Kalman filter or others. The reason is that they can be very fast and since a real-time performance was difficult to achieve, they were amongst the first methods to be used. The limiting factor is that they only process linear functions which the robot localization is not (although it can be minimized by carefully designing the functions involved or overcome with iterative variants of the extended Kalman filter, see the overview by Havlík et al. [13]). The function needs to be linearized, causing linearization errors.

A famous algorithm from the category of iterative minimization techniques is *bundle adjustment* (a modern overview was done by Triggs [14]), which is used in the Structure from Motion, where the 3D structure and corresponding camera poses are recovered from a set of images. As the computational power of computers grew, many variants of bundle adjustment emerged, typically involving a subset of the last few camera frames called the keyframes. They are typically more computationally heavy but promise higher accuracy, as they can relinearize the measurement function many times to obtain a more accurate result.

1.1.3 Sensors

A very distinctive categorization is according to the sensors used.

In theory, by integrating data from gyroscope and accelerometer, the position and orientation estimate can be obtained. This approach is not viable as the IMU noises are amplified by the integration, leading to rapid position drift after just a few seconds. This happens even with high-quality sensors. It does not mean that these sensors cannot be used for localization, they just need to be complemented with other sensors.

A large set of the methods is, to some extent, utilizing the camera. It is cheap, accurate and provides rich information about the environment. Both one (monocular) or multiple (stereo) cameras can be used. A stereo camera allows to easily estimate the depth and can often be found on scientific instruments.

Methods involving just a camera are limited in their robustness. First, the scale is not observable by a vision-only system. A big object far away from the camera will appear the same as a smaller version of the same object close to it. It is possible to recover the relative scale but not the absolute one[1]. The second problem is the relative motions ambiguity. It can be demonstrated by a camera mounted in a train, looking out of the window and seeing another train. When one of the trains starts moving, the system cannot distinguish which one it is. This issue arises every time when a moving object is present in the scene. Many techniques exist to minimize this issue but it will always be present, limiting the real-world robustness.

An interesting combination is when a camera (either monocular or stereo) is mounted together with an IMU as both sensors are ”orthogonal” in some sense.

[1]For example a monocular SLAM method can build a map with consistent scale across the map, meaning that the map can be converted to the meters simply by scaling all dimensions by a single number s. The absolute scale cannot be recovered from such system, that is the number s cannot be estimated without some external measurement.

IMU allows to determine the absolute scale and can distinguish whether it is the camera or its surroundings that are moving. From the other side, the information from the camera can help to estimate the noise parameters of the IMU and correct the position drift. Because of this and also because both sensors are very affordable and omnipresent, this combination is very popular.

An increasingly popular sensor is Light Detection and Ranging (LIDAR). It works by emitting one or more beams of laser light and measuring the time it takes for the reflection to reach the sensor. The time is then converted to a distance. LIDARs have the advantage of being very accurate, up to a few millimeters on 50 meters of distance. They work across many illumination conditions both indoor and outdoor. Because they are active, in the sense that they are actively emitting the light which they are detecting, they can even work at night. On the other hand, they are very expensive and not very wide-spread yet.

Another common sensor is GPS. In fact, many real-time kinematics (RTK) GPS sensors utilize a sensor fusion under the hood, using accelerometer measurements to suppress the noise of raw GPS measurements and to provide the output at a higher frequency.

In general, any sensor providing the estimate of position, velocity, acceleration, orientation, rotation velocity or anything correlated to these can be used. It could be a barometer, magnetometer, sun sensor, gimbaled gyroscope, rotation encoder from the wheels (wheel odometry), convolutional neural network processing the camera images, depth-sensing structured light cameras, event cameras[2] or even an amperemeter measuring the current flowing into the motors of a quadcopter.

1.1.4 Sparse and dense, direct and indirect methods

For the methods involving the camera, this is an important subdivision determined by the approach to processing the image information.

Sparse methods first extract distinct features from the image and only work with these features. The most commonly detected features are corners (i.e. a small area or pixel with rapid change in intensity in all directions) or edges (similar to corners but the change is only in one dimension). In comparison, dense methods usually work with information from all pixels or from their subset (semi-dense methods).

When processing the information, keypoints from two or more images need to be matched to create a multi-view constraint. This can be done either directly or indirectly for both sparse and dense techniques. Direct methods match the keypoints based on the pixel intensities directly, by comparing patches around the points of interest. Indirect methods first build vectors (called descriptors) describing each patch. When matching keypoints from two images, a metric based on their descriptors is used (commonly Hamming or Euclidian distance). The direct methods can be significantly simpler and faster but they are not invariant to stronger viewpoint and illumination changes. They are commonly used for video feed processing where these effects tend to be less significant.

[2]Instead of providing an image as a snapshot in time, event cameras provide a stream of events. Each event is a change of the intensity of particular pixel at given time. Although processing of such information is less straight-forward, they enable camera-based localization even in situations where high-speed cameras fail, such as scenarios with rapid movements.

1.1.5 Keyframe or non-keyframe methods

Another subcategory of camera-based methods is the usage of keyframes. The idea is that not all the frames from the camera contribute the same amount of information. Therefore, it is safe to leave some of them out and only keep the important ones - keyframes. Keyframe-based techniques are commonly used with iterative minimization approaches, mostly for speedup. Filtering-based methods usually use the all the frames as they are not that computationally expensive.

1.2 An overview of localization methods

1.2.1 EKF-SLAM

Extended Kalman Filter (EKF)-SLAM is one of the first techniques for robot localization. A modern overview of the topic was provided by Durrant-Whyte et al. [15]. It is an EKF-based SLAM technique that includes the current camera pose and a map in the filter state. The map contains position estimates of all the features that have been seen. The EKF-SLAM can provide an accurate estimate of the current location but only inside of a limited area. When exploring larger areas, the filter state size grows quite fast. The computational complexity of the filter is at least quadratic in the state size, therefore it quickly becomes computationally infeasible. Nowadays, it is mostly used for research purposes as it allows fusion with various sensors, including monocular and stereo cameras, IMUs and many others. It is also often used as a theoretical tool when proving properties of other filtering-based techniques [16, 17, 18].

1.2.2 MSKCF

Multi-State Constraint Kalman Filter (MSCKF) [19, 20] is a filtering-based odometry, utilizing the IMU and a monocular camera. It is similar to the EKF-SLAM but instead of including all the features in the state, they include a rolling window of the last few camera positions and rotations (poses). The authors showed that this filter can achieve very accurate localization, even on a long distance and without a map.

This work was further improved based on theoretical analyses of the underlying model [21, 18, 22] and extended by more accurate sensor modeling [23, 24, 25] and by utilizing different kinds of Kalman filter [26, 27].

1.2.3 ROVIO

ROVIO [28, 29] is similar to MSCKF and utilizes the same kind of sensors. The main difference is that it is a direct method which works with the pixel intensities directly. Also, it uses the robocentric approach, where the position of all the features is expressed in the current camera frame, instead of fixed to the world. This solves some of the known issues (observability properties) of the original MSCKF.

1.2.4 OKVIS

OKVIS [30] is a keyframe-based iterative minimization odometry, using one or more cameras. The author performed a thorough comparison to MSCKF, showing that it outperforms it in terms of accuracy but is more computationally intensive.

1.2.5 VINS-Mono

VINS-Mono [31, 32], is a keyframe-based iterative minimization SLAM. It uses a monocular camera in sparse, indirect way. Its suthors claim that compared to OKVIS, this method is more accurate. Another thing that stands out about VINS-Mono is that its authors provided a public implementation running on a commodity device (iPhone).

1.2.6 PTAM

Although being limited to small working area, PTAM [33] is a breakthrough work. It was the first one to employ a schema where one CPU thread is responsible for fast camera tracking and others for place recognition and map optimization. That enables very responsive position estimates even though a complex, long-running optimization is happening under the hood.

1.2.7 ORB-SLAM

ORB-SLAM [34] is following the same multi-threaded pattern introduced in PTAM. It is a visual-only SLAM, based on ORB features that are performance optimized, making it run in real-time. This method is very widely known and used, likely because it is open-source, easy to set up and has relatively low hardware requirements.

1.2.8 LSD-SLAM

LSD-SLAM [35] is another visual-only SLAM but utilizing direct, semi-dense image processing. Although it processes a large amount of data, it is able to run in real-time without using the GPU. Despite of being less accurate, this method is also very widely used. The main factor contributing to its success might be that since it is semi-dense method, the underlying map is easy to read and visually pleasing.

1.2.9 DTAM

DTAM [36] is a dense SLAM, utilizing a single monocular camera. Similar to PTAM, it is only able to cover a small area and it needs to run on a GPU. The output map is very visually informative as it is displayed as a colored surface instead of a set of points. Thanks to the direct approach, this method is very robust to motion-blur of the image or camera defocus.

1.2.10 SVO

SVO [37] is a sparse, direct visual method, capable of running in either odometry or SLAM mode. It was designed with the speed in mind so it uses model-based image alignment instead of relatively expensive feature-point matching. It works by directly estimating the pose of a new image, using the error defined directly by the pixel intensities. According to the paper, this allows for accurate position tracking at 300 FPS on a desktop computer.

2. Proposed solution

This chapter discusses the algorithm that will be implemented and the options that are available to achieve the goal which has been set.

2.1 Proposed solution

The goal of this work is to provide a technique that would be capable of lifelong localization of robots without any assumption on the working area, especially its boundedness. This practically rules out usage of the SLAM-based techniques. They can provide a reliable lifelong localization but since the size of the map grows at least linearly with the working area of the robot, they cannot provide a truly lifelong localization when, for example, a car travels a long distance from one place to another, without revisiting the same places. Also, in such a scenario a SLAM-based technique technically degrades to an odometry method because it cannot reuse the map. Hence the map building efforts have diminishing returns in such a scenario.

On the other hand, odometries are able to operate in an unbounded area for arbitrary long periods of time but the estimated position of the robot will inevitably drift in time. This drift might be very slow for some methods but it will, in principle, be still present, unless corrected by external measurements.

The chosen approach to fulfill the requirements is to use one of the accurate odometry algorithms and fuse it with the GPS measurements. This will lead to an algorithm that is capable of tracking the position accurately when the robot operates indoors or in any other GPS-denied areas, and when the GPS is available, it will use it to correct the position estimate. This will produce a system that is capable of accurate localization with bounded position drift.

The assumption of having the GPS measurements available, at least sparsely, is not very strong in this case. When the robot only operates indoors, where no GPS is available, it can use one of the SLAM-based techniques as they are capable of mapping quite large areas, likely even the largest buildings in the world. On the other hand, if the working area is truly unlimited, it will at least partially operate outdoors, where the GPS measurements are available, and thus keeping the position drift bounded.

2.2 Odometry choice

From the list of the available methods presented in chapter 1, the SLAM-based techniques are filtered out. Also the vision-only techniques are filtered out as they suffer from loss of robustness in some scenarios. The remaining methods are MSCKF, ROVIO and OKVIS. OKVIS is a very promising method in terms of accuracy and possible future research. Despite that, it was ruled out because of its high computational demands. Even though it is likely able to run in real time on a commodity hardware, it would drain the battery fast and leave out only little resources for other applications. That would also limit its possible applications. This decision is very dependent on the current status of the technology and should

be revisited in the future.

The remaining methods are MSCKF and ROVIO, both of which are similar. It was decided to proceed with MSCKF as it has more research activity surrounding it. This makes it easier to understand the inner workings of the algorithm which is necessary in order to implement further extensions.

The accuracy of MSCKF is good enough for extended periods of time but definitely not for lifelong localization. By its nature, the position estimation error will slowly drift in time indefinitely.

3. Preliminaries

3.1 Notation

This work follows the notation used in [19]. A simple overview is presented in this section. Note that there are some ambiguities in this notation.

Scalars are lower case
Time t

Vectors are bold
Position in 3D $\mathbf{p}$
Position of feature f in 3D $\mathbf{p}_f$

Matrices are uppercase bold
General rotation matrix $\mathbf{R}$

Frames of reference are uppercase with optional curly braces
Global frame G, $\{G\}$

Frames of reference of given points are upper prescript
Point $\mathbf{p}$ expressed in body frame $\{B\}$ ${}^{B}\mathbf{p}$
Position of body frame expressed in global frame ${}^{G}\mathbf{p}_{B}$

Rotation matrices have lower and upper prescript
Rotation matrix from body to global frame ${}^{G}_{B}\mathbf{R}$

Rotation matrices and quaternions are used interchangeably
Rotation matrix from body to global frame as quaternion ${}^{G}_{B}\mathbf{q}$

Continuous derivations by time are denoted by dot
Derivation of velocity by time $\dot{\mathbf{v}}$

Estimates are denoted by hat
Estimate of the filter state $\hat{\mathbf{x}}$

Differences are denoted by Δ
Time difference between step $k+1$ and k $\Delta t = t_{k+1} - t_k$

Orientation differences are denoted by δ
Perturbation of angle θ $\delta\theta$
Perturbation of quaternion $\mathbf{q}$ $\delta\mathbf{q}$

Error state is denoted by tilde
Error state of the filter $\tilde{\mathbf{x}}$

Predictions for different time steps are denoted by subscript
Filter state at step $k+1$ predicted from step k $\mathbf{x}_{k+1|k}$

Quaternion multiplication is denoted by $\otimes$
Product of quaternions $\mathbf{q}_1, \mathbf{q}_2$ $\mathbf{q}_1 \otimes \mathbf{q}_2$

Reassignment of a variable is denoted by $\leftarrow$
Reassignment of variable x to x_{new} $x \leftarrow x_{\text{new}}$

3.2 Sensors

3.2.1 Gyroscope

A gyroscope measures the rotational velocity $\boldsymbol{\omega}$, with units of *rad/s*.

Earlier gyroscopes were realized as a spinning mass, suspended such that it can rotate freely around all axes. As a consequence of the law of conservation of angular momentum, such gyroscope keeps its orientation regardless of the rotation of the frame. This can be used to directly observe the rotation relative to a fixed frame of reference.

Later, other types of gyroscopes emerged. Instead of measuring the orientation directly, they measure the rotation velocity instead. By integrating the rotation velocity over some period of time, an absolute orientation relative to some frame of reference can be determined. Examples of such sensors are ring laser gyroscope (RLG) or fibre optic gyroscope (FOG). They are both based on the same principle. Two coherent[1] beams of light with a phase shift of 180° travel the same trajectory in the opposite direction. If the body rotates around a specific axis, one of the beams has to travel a slightly longer distance. The beams are then merged together. If they had both traveled the same distance, they would interfere, canceling each other. Otherwise, an interference pattern emerges and enables to calculate the rotation rate. Such a gyroscope only measures the rotation rate around a single axis, so they are usually coupled in a form of three orthogonal sensors.

Although RLG and FOG are very accurate, they are expensive and not very small. Nowadays, a different kind of gyroscope is used in the majority of applications. It is called vibrating structure gyroscope (VSG). When an object vibrates, it tends to vibrate in the same plane, even if its support rotates. When the gyroscope rotates, the vibrating object applies a force on the support. The rotation rate can be determined by measuring this force. VSGs are typically manufactured as Microelectromechanical systems (MEMS) and can be very cheap. Also, they are solid state, meaning that there is no rotating body, which increased their reliability. For the same reason as RLG and FOG, they are usually used in a combination of three orthogonal sensors.

Many other types of gyroscopes exist, based on various physics phenomena. For the purpose of this work, MEMS VSG is considered, even though any other type could be used.

The perfect gyroscope would measure the rotation rates exactly. No device is perfect though so, in reality, the measurement is corrupted by many sources of error. One of the commonly used models is

$$ {}^{B}\boldsymbol{\omega}_m(t) = {}^{B}\boldsymbol{\omega}(t) + \mathbf{b}_g + \mathbf{n}'_g \tag{3.1} $$

where ${}^{B}\boldsymbol{\omega}_m(t)$ is the measurement at time t and ${}^{B}\boldsymbol{\omega}(t)$ is the true rotation rate at that time. $\mathbf{b}_g$ is a 3×1 vector of measurement bias. Its value slowly evolves in time and it is modeled as a random-walk where $\dot{\mathbf{b}}_g \sim \mathcal{N}(\mathbf{0}, \mathbf{Diag}(\mathbf{n}_{wg}))$. $\mathbf{n}_g$ is a noise vector of the same size and it is independent for each measurement

$$ \mathbf{n}'_g \sim \mathcal{N}(\mathbf{0}, \mathbf{Diag}(\mathbf{n}_g)) \tag{3.2} $$

[1] Two light beams are coherent if they have the same frequency and a constant phase shift.

Among other imperfections, the three gyroscopes that form the sensor might not be mounted perfectly orthogonally, they might have scale imperfections or even be influenced by an acceleration. To consider these properties, a more sophisticated model can be used

$$^{B}\boldsymbol{\omega}_m(t) = \mathbf{T}_g {}^{B}\boldsymbol{\omega}(t) + \mathbf{T}_a {}^{B}\mathbf{a} + \mathbf{b}_g + \mathbf{n}'_g \tag{3.3}$$

where $\mathbf{T}_g$ models sensor misalignment and scale errors and $\mathbf{T}_a$ models the influence of the acceleration $^{B}\mathbf{a}$[2].

Another factor that can influence the performance of a gyroscope is the temperature. Gyroscopes are commonly calibrated by the manufacturer at a specific temperature. Some, usually more expensive models are calibrated for a wide range of temperatures. The temperature can also be used in the measurement model in order to provide accurate measurements even with low-grade sensors.

3.2.2 Accelerometer

An accelerometer measures the linear acceleration applied to the body.

In contrast to gyroscopes which use many different phenomena to measure the rotation rate, the majority of the accelerometers are based on the same underlying principle. An object of a known mass is suspended on a spring and it is placed on a piezoelectric material. When acceleration is applied to the body, the object is displaced, producing a force to the piezoelectric material. This force is then measured and converted to an acceleration measurement.

A single accelerometer measures the acceleration in a single direction. So they are commonly packaged as three orthogonal devices.

Accelerometers are also subject to many imperfections. A simpler way of modeling them is to consider bias and measurement noise, similar to the gyroscope:

$$^{B}\mathbf{a}_m(t) = {}^{B}\mathbf{a}(t) + \mathbf{b}_\mathbf{a} + \mathbf{n}'_a \tag{3.4}$$

where $^{B}\mathbf{a}_m(t)$ is the measurement at time t and $^{B}\mathbf{a}(t)$ is the true acceleration applied to the sensor. The measurement is corrupted by a bias $\mathbf{b}_a$, modeled as a random-walk such that $\dot{\mathbf{b}_a} \sim \mathcal{N}(\mathbf{0}, \mathbf{Diag}(\mathbf{n}_{wa}))$. Another source of error is a measurement noise $\mathbf{n}'_a$ which is independent for each measurement and

$$\mathbf{n}'_a \sim \mathcal{N}(\mathbf{0}, \mathbf{Diag}(\mathbf{n}_\mathrm{a})) \tag{3.5}$$

An alternative model, especially suitable for low-grade accelerometers is

$$^{B}\mathbf{a}_m(t) = \mathbf{T}_a {}^{B}\mathbf{a}(t) + \mathbf{b}_\mathbf{a} + \mathbf{n}'_a \tag{3.6}$$

where the matrix $\mathbf{T}_a$ models sensor scale and misalignment errors. The misalignment is caused by the sensors not being mounted exactly orthogonally. The acceleration it the direction of a single axis is them partially measured by other accelerometers.

[2]The acceleration influence on rotation rate measurements is sometimes called g-sensitivity, as acceleration can be measured in units g.

3.2.3 Monocular camera

Cameras are sensors with very rich information output. Combined with the fact that they are relatively cheap and widespread, there are many applications for them. There is a whole area of computer vision that studies cameras and their possible applications. For the purpose of this work, there are few relevant subjects that need to be discussed.

Pinhole camera

A suitable way of modeling a camera depends on its field of view. Some cameras have close to 180° field of view. They are referred to as fisheye cameras and they need to be modeled in a special way. Consumer electronic cameras tend to have the field of view around 60°. These cameras can be modeled using a pinhole camera model.

A pinhole camera is a camera without any lens but with a small aperture - a pinhole. The light from the scene passes through the pinhole, projecting the scene upside down onto the image plane located behind the pinhole. Modern cameras use one or more lenses to achieve the same effect but they can still be modeled using this simplified model.

Since the real image is captured on the image plane upside down, it is easier to work with a *virtual image plane* for the purpose of the analysis. It is a plane located at the same distance from the aperture as the *image plane* but on the other side of it. The whole situation is depicted in figure 3.1. The image on the virtual image plane has the same geometrical properties as the image on the image plane but it is not upside down.

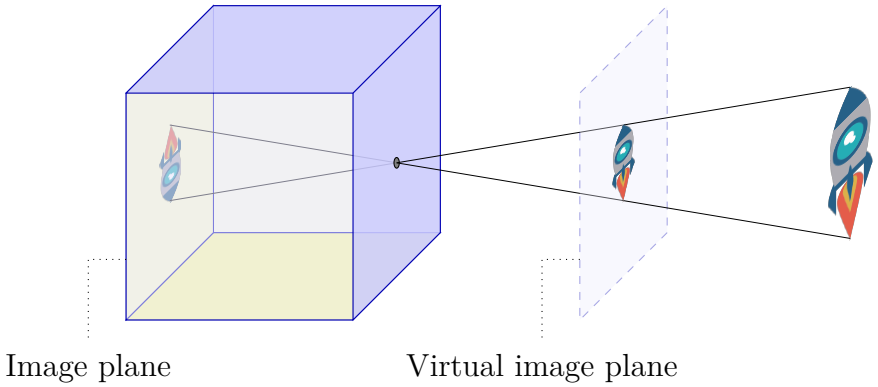


Figure 3.1: Pinhole camera model with a virtual image plane.

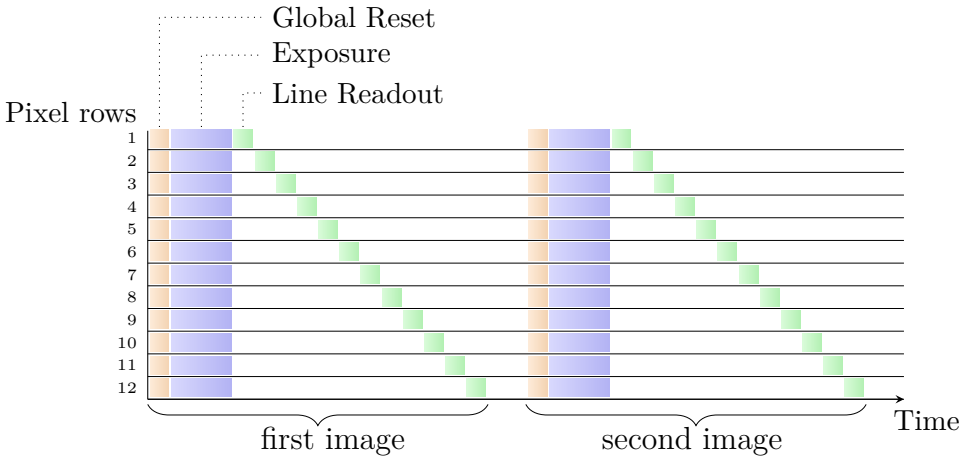
Rolling shutter cameras

During the image capturing process, the majority of time is spent by sending the image from the individual pixels to the processor. The problem gets amplified on modern cameras that can have up to 100 MPx (100,000,000 pixels). When the image is being transferred, pixels cannot accumulate more light, which limits the maximum exposure time and degrades the performance in low-light conditions.

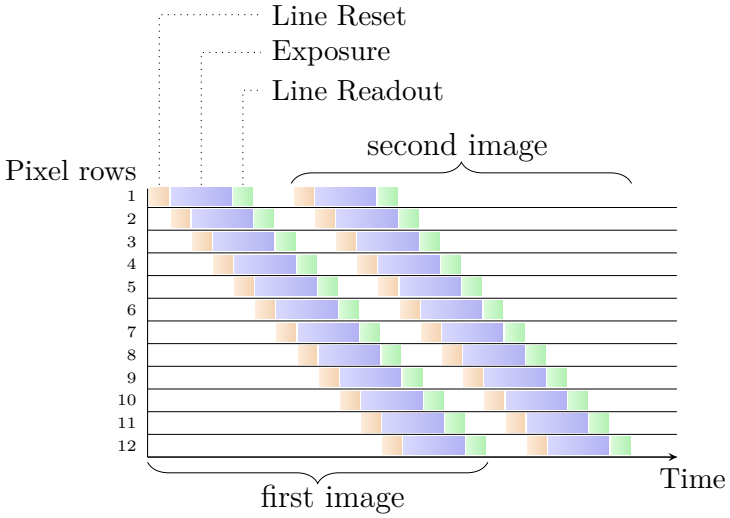
A common way of overcoming this issue is *rolling-shutter* (fig. (3.2b)). The individual rows are read one by one, maximizing the time each pixel can use for

the exposure. The time during which a single image is read is called a *readout time*. It can be calibrated by taking a video of a flashing light at a known frequency.

Cameras that expose the whole image at the same time are called *global-shutter* cameras (fig. (3.2a)).



(a) In global-shutter cameras, the whole image is exposed at the same time.



(b) Rolling-shutter cameras expose individual lines.

When the captured scene is stationary, both cameras produce the same image. The difference becomes noticeable when either some object in the scene or the camera is moving. Depending on the type of the movement, for example, vertical lines might appear curved or skewed. The effect can be seen in fig. (3.3) and (3.4).

3.3 Computer vision

3.3.1 Point feature projection

A camera is projecting a scene onto a plane, forming the image. Given a 3D position of point A and camera parameters, it is possible to determine a 2D position where the point A would be projected.

Figure 3.3: Rolling-shutter camera is stationary [38].

Figure 3.4: The same camera follows the train [38].

A focal length f of a camera is the minimum distance from the aperture to the (virtual) image plane, measured in meters. If not occluded, point A with coordinates ${}^{C}\mathbf{p}_A = \begin{bmatrix} X & Y & Z \end{bmatrix}$ relative to the camera will be projected onto a point $\mathbf{z}_a = \begin{bmatrix} x & y \end{bmatrix}$ such that

$$x = f\frac{X}{Z} \qquad\qquad y = f\frac{Y}{Z} \tag{3.7}$$

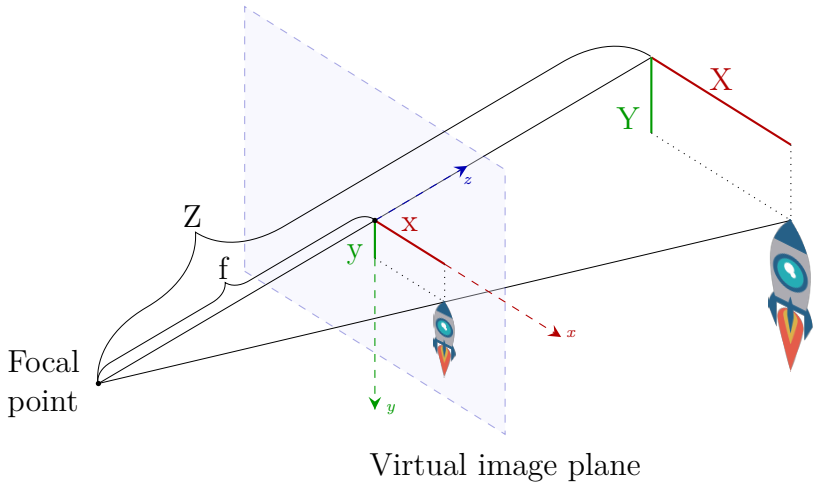
The situation is depicted in fig. (3.5a). This coordinate system of an image is sometimes referred to as an x-y coordinate system. Image coordinates are also conveniently measured in pixels - a UV coordinate system. They can be easily converted to each other but there are some aspects that need to be considered. First, not all cameras have square pixels. Assuming that the camera has pixels of size $d_x \times d_y$ (in meters), its focal length measured in pixels would be $f_x = f/d_x$, $f_x = f/d_x$. Second, pixels are indexed from the upper left corner, instead of the center. The projection of a 3D point into the image, measured in pixels is then

$$u = f_x\frac{X}{Z} + o_x \qquad\qquad v = f_y\frac{Y}{Z} + o_y \tag{3.8}$$

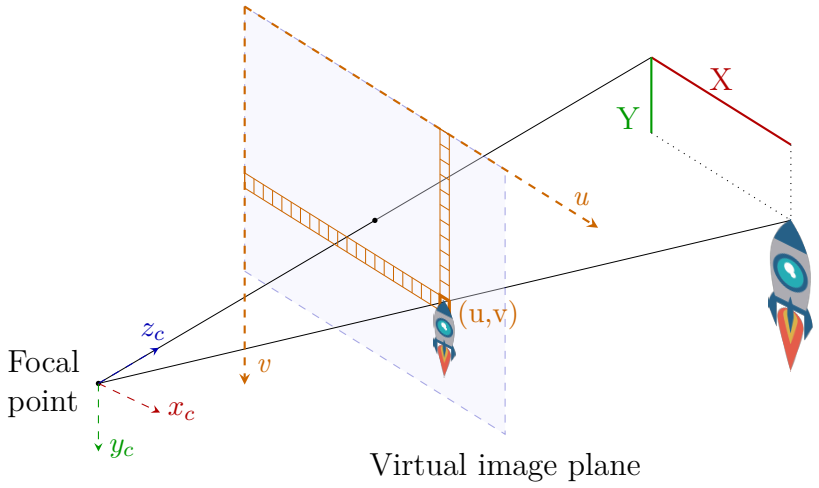
where (o_x, o_y) is the origin of the x-y coordinate system measured in pixels, as can be seen in fig. (3.5b). Equivalently in matrix notation with the model being labeled as a function $\mathbf{h}$

$$\mathbf{z}_A = \mathbf{h}\left({}^{C}\mathbf{p}_A\right) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} X/Z \\ Y/Z \end{bmatrix} + \begin{bmatrix} o_x \\ o_y \end{bmatrix} \tag{3.9}$$

Real-life cameras usually suffer from image distortion caused mostly by lens artifacts. To take them into account, a radial distortion with three parameters (k_1, k_2, k_3) and a tangential distortion with two parameters (t_1, t_2) can be used,



(a) Projected point, measured in normalized coordinates.



(b) The same point, measured in pixel coordinates.

Figure 3.5: A point from the scene is projected onto the (virtual) image plane using similar triangles. There are infinitely many similar triangles (one for each value of Z) that project onto the same point, which leads to the loss of the scale information.

extending the model to[3]

$$\mathbf{h}\left({}^{C}\mathbf{p}_{A}\right) = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \left(d_r \begin{bmatrix} x \\ y \end{bmatrix} + d_t \right) + \begin{bmatrix} o_x \\ o_y \end{bmatrix} \tag{3.10}$$

$$d_r = 1 + k_1 r + k_2 r^2 + k_3 r^3$$

$$d_t = \begin{bmatrix} 2xyt_1 + (r + 2x^2)\, t_2 \\ 2xyt_2 + (r + 2y^2)\, t_1 \end{bmatrix}$$

$$x = \frac{X}{Z}, \quad y = \frac{Y}{Z}, \quad r = x^2 + y^2$$

[3]This camera model is also used by a widespread library OpenCV [39], making it easy to calibrate and work with.

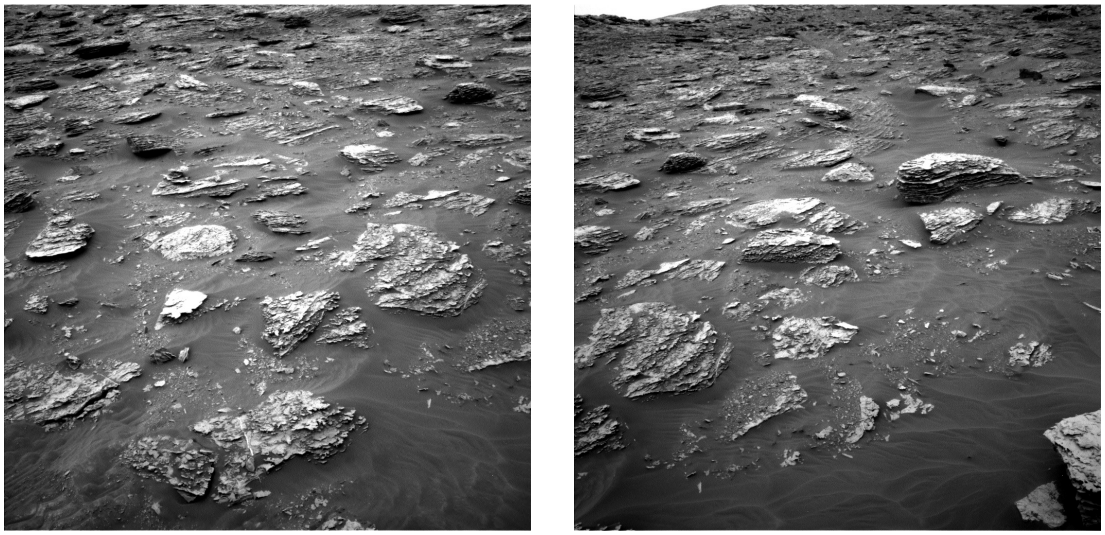


Figure 3.6: These images are from Mars, captured a few meters apart. It would be extremely difficult for a human to match the common part. A SIFT algorithm can find 323 matching points. The result can be seen in figure 3.9 on page 25. The images were captured by the Curiosity rover at Sol 2087. They are from NASA Jet Propulsion Laboratory [40].

3.3.2 Feature tracking

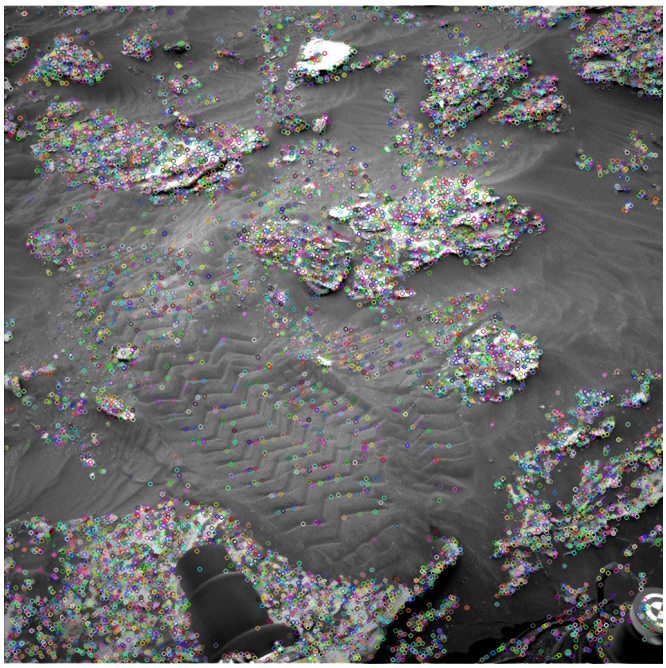
Feature tracking is a very well established area of computer vision. It uses multiple images to track a point - feature. Being able to track a feature in multiple images is crucial for applications like visual and visual-inertial odometry, structure from motion, stereo vision, camera calibration, panorama stitching and others.

Over the years, many methods of feature tracking emerged. A choice of the algorithm is very important as it is a computationally intensive task and methods differ in capabilities widely. For example, some feature matching schemas are able to match the points from two images with arbitrary rotation (they are then called rotation covariant features) and some others are not. Other criteria include image noise, illumination change, viewpoint change, motion blur or feature scale.

A common choice of feature matching pipeline is based on the SIFT algorithm [41]. It starts by extracting the feature points from given image. A good feature point should be detected in the image even when observed from different viewpoint, under different lighting conditions or with the different camera. Example of SIFT features detected in an image is shown in figure 3.7a. Next, a descriptor vector is found for each feature point. In case of SIFT, the descriptor is a vector of 128 real numbers and it is based on the histogram of oriented gradients in the patch around the feature. The closer two vectors are, the more likely they represent the same point. The distance of two descriptors is measured by a standard L2 norm. An example result of SIFT matching can be found in figure 3.9.

SIFT is commonly used in structure-from-motion pipelines, where robustness is the foremost concern and real-time performance is not required. The extraction step for each image at the figure 3.9 took about 416 ms and matching took 4.32 s, utilizing four cores of a modern CPU.

In real-time applications, other kinds of features are used. Among the fastest



(a) SIFT features found in the image.

(b) First 40 descriptors visualized as an image. Every descriptor is a single line of the image.

ones is FAST [42]. The author described the algorithm as follows: "If $\geq N$ contiguous pixels in a Bresenham circle of radius r around a centre pixel p are all brighter than p by some threshold or all darker than p by some threshold, then there is a feature at p.". Such a circle is shown in figure 3.8. The exact thresholds are determined by machine learning. The simplicity of this algorithm is the key to its fast execution time. The feature detection step can be done on CPU in as fast as 1 ms, allowing for real-time performance. It is important to mention that FAST algorithm is only a feature detector. In order to match the features, other algorithms need to be used on top of it.

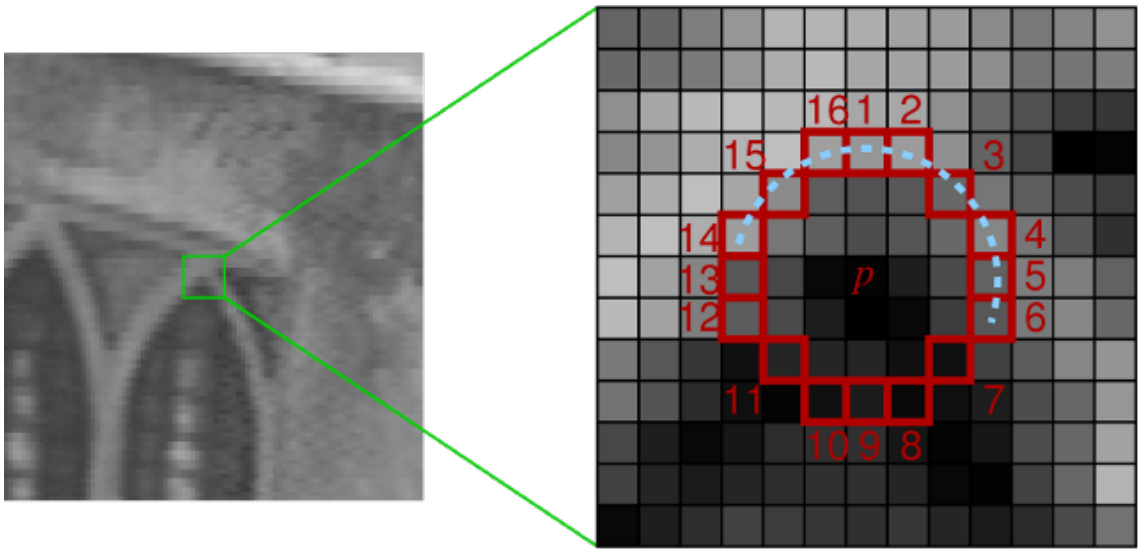


Figure 3.8: A FAST algorithm only looks at a few pixels, allowing for very fast extraction times.

A popular feature detector and descriptor based on FAST is ORB [43], a fast and robust feature matching algorithm. It is a base of very popular ORB-SLAM [44]. In contrast to SIFT, it uses a string of bits as a descriptor, with the distance being measured by Hamming distance. The Hamming distance can be computed very fast on modern CPUs as it is computed by applying a XOR operation combined with counting the number of 1s in the result. Even though ORB features are suitable for most of the visual-inertial odometries, there is another, faster approach available.

When an extra information is known about the images, it is possible to develop a faster and more robust algorithm. One of the common tasks is to track features in a video stream. Obviously, all the frames are captured by the same camera, leaving out the need to estimate the camera intrinsics for every frame if needed for image matching pipeline. Also, since the videos are often recorded at 24 Hz to 60 Hz, the camera only has a little time to move in between two consecutive frames. Therefore it is safe to assume that a feature is located in a similar location in the next frame. The higher the frame rate, the more similar two consecutive frames are.

A famous well-studied algorithm for tracking features in consecutive frames is *KLT tracker*[4] [45, 46]. Given two images $\mathbf{F}$ and $\mathbf{G}$ and a feature point $\mathbf{x}$ from

[4]KLT tracker is an example of sparse optical flow algorithm which is a largely studied area.

image $\mathbf{F}$, the goal is to find the corresponding point in image $\mathbf{G}$. An image can be viewed as a function of a single 2D variable, taking the pixel coordinates and returning the pixel intensity. When non-integer coordinates are supplied, a 2D interpolation is carried out. Starting from disparity vector $\mathbf{h} = \mathbf{0}_{2\times1}$, KLT tracker tries to minimize the error function

$$\mathbf{E} = \sum_{\mathbf{x}\in R} \left(\mathbf{F}(\mathbf{x}+\mathbf{h}) - \mathbf{G}(\mathbf{x})\right)^2 \tag{3.11}$$

where R is a patch around the pixel, usually a square of size 3×3 or 5×5. The linear approximation of $\mathbf{F}$ is

$$\mathbf{F}(\mathbf{x}+\mathbf{h}) \simeq \mathbf{F}(\mathbf{x}) + \mathbf{h}\left(\frac{\partial\,\mathbf{F}(\mathbf{x})}{\partial\,\mathbf{h}}\right)^T \tag{3.12}$$

The minimum of the error function can be found by computing the partial derivatives and comparing them to zero.

$$\mathbf{0} = \frac{\partial\,\mathbf{E}}{\partial\,\mathbf{h}} \simeq \frac{\partial}{\partial\,\mathbf{h}} \sum_{\mathbf{x}\in R} \left(\mathbf{F}(\mathbf{x}) + \mathbf{h}\left(\frac{\partial\,\mathbf{F}(\mathbf{x})}{\partial\,\mathbf{x}}\right)^T - \mathbf{G}(\mathbf{x})\right)^2 \tag{3.13}$$

$$= 2\sum_{\mathbf{x}\in R} \left(\mathbf{F}(\mathbf{x}) + \mathbf{h}\left(\frac{\partial\,\mathbf{F}(\mathbf{x})}{\partial\,\mathbf{x}}\right) - \mathbf{G}(\mathbf{x})\right)\left(\frac{\partial\,\mathbf{F}}{\partial\,\mathbf{x}}\right) \tag{3.14}$$

$$\Rightarrow \mathbf{h} \simeq \left(\sum_{\mathbf{x}\in R} (\mathbf{G}(\mathbf{x}) - \mathbf{F}(\mathbf{x}))\left(\frac{\partial\,\mathbf{F}}{\partial\,\mathbf{x}}\right)\right)\left(\sum_{\mathbf{x}\in R}\left(\frac{\partial\,\mathbf{F}}{\partial\,\mathbf{x}}\right)^T\left(\frac{\partial\,\mathbf{F}}{\partial\,\mathbf{x}}\right)\right)^{-1} \tag{3.15}$$

This leads to one gradient descent step of a single feature matching. More iterations (usually 5 to 30) are needed until the termination criterion is met. A final vector $\mathbf{h}$ then represents the displacement of the feature in the next image. Because all the derivative terms can be precomputed once and reused for all the update steps of all features, KLT tracker is well-suited for real-time applications.

3.3.3 Feature point triangulation

When a feature point is tracked in multiple frames and position and orientation of the camera at the time the image was taken is known, the position of a feature can be estimated using triangulation. This procedure is essential for most of the computer vision algorithms.

In the simplest case, it is enough to take two cameras observing a feature. Each camera then defines a ray in 3D on which the feature must be. In theory, the intersection of the rays gives the position of the feature. In practice, those rays never intersect because of the noises involved in both feature tracks and camera pose estimates.

The most common way of performing a triangulation is to define an error function which takes an estimate of feature's position and returns how well it corresponds to the measurement. The error can be then minimized using any optimization algorithm, producing the triangulated position.

A popular error function is the feature reprojection function[5]. The idea of the algorithm is to express the point in each camera's coordinates, use the camera

[5]Other option might be epipolar point transfer [47].

projection function from eq. (3.11) with the error being the difference between this reprojected point and the observed position of the feature in that camera.

More formally, the feature was observed from cameras $C_i, i \in 1, \ldots, n$. Each camera has a known position ${}^{G}\mathbf{p}_{C_i}$ and rotation ${}^{C_i}_{G}\mathbf{R}$, relative to some fixed global frame $\{G\}$. The i-th camera observed the feature at 2D coordinates $\mathbf{z}_i$ (pixels). The goal is to estimate ${}^{G}\mathbf{p}_f$, the position of the feature in global frame.

To be able to apply the camera projection function, ${}^{G}\mathbf{p}_f$ needs to be expressed in the camera's frame, obtaining ${}^{C_i}\mathbf{p}_f$. It can be computed as

$$ {}^{C_i}\mathbf{p}_f = {}^{C_i}_{G}\mathbf{R}\left({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_{C_i}\right) \tag{3.16} $$

Then the reprojection function for the i-th camera $E_i({}^{G}\mathbf{p}_f)$ is

$$ E_i({}^{G}\mathbf{p}_f) = \frac{1}{2}\left\|\mathbf{z}_i - \mathbf{h}\left({}^{C_i}\mathbf{p}_f\right)\right\|^2 \tag{3.17} $$

and the final error function is

$$ E({}^{G}\mathbf{p}_f) = \frac{1}{2}\sum_{i=1}^{n} E_i({}^{G}\mathbf{p}_f) \tag{3.18} $$

$$ = \frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{z}_i - \mathbf{h}\left({}^{C_i}_{G}\mathbf{R}\left({}^{G}\mathbf{p}_f - {}^{G}\mathbf{p}_{C_i}\right)\right)\right\|^2 \tag{3.19} $$

Although such approach works well in most of the situations and is commonly used, it can be further improved. The problem is with features that are located far from all the cameras. The partial derivatives along each direction will differ widely, causing numerical instabilities. For this reason, an *inverse-depth parametrization* [48] is used. It expresses the position of a feature in the camera frame as a ray and an inverse value of feature's distance from the camera (depth). Instead of ${}^{G}\mathbf{p}_f$, the result of inverse-depth parametrization is ${}^{C_1}\mathbf{p}_f$ which can be easily converted to the global frame. Starting from the usual transformation

$$ {}^{C_i}\mathbf{p}_f = {}^{C_i}_{C_1}\mathbf{R}\left({}^{C_1}\mathbf{p}_f - {}^{C_1}\mathbf{p}_{C_i}\right) \tag{3.20} $$

$$ = {}^{C_i}_{C_1}\mathbf{R}\,{}^{C_1}\mathbf{p}_f + {}^{C_i}\mathbf{p}_{C_1} \tag{3.21} $$

$$ = {}^{C_i}_{C_1}\mathbf{R}\begin{bmatrix} {}^{C_1}X \\ {}^{C_1}Y \\ {}^{C_1}Z \end{bmatrix} + {}^{C_i}\mathbf{p}_{C_1} \tag{3.22} $$

$$ = {}^{C_1}Z\left({}^{C_i}_{C_1}\mathbf{R}\begin{bmatrix} {}^{C_1}X/{}^{C_1}Z \\ {}^{C_1}Y/{}^{C_1}Z \\ 1 \end{bmatrix} + \frac{1}{{}^{C_1}Z}{}^{C_i}\mathbf{p}_{C_1}\right) \tag{3.23} $$

$$ = {}^{C_1}Z\left({}^{C_i}_{C_1}\mathbf{R}\begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} + \gamma{}^{C_i}\mathbf{p}_{C_1}\right) \tag{3.24} $$

$$ = {}^{C_1}Z\mathbf{g}_i(\boldsymbol{\theta}), \qquad \boldsymbol{\theta} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T \tag{3.25} $$

where

$$ \alpha = \frac{{}^{C_1}X}{{}^{C_1}Z} \qquad \beta = \frac{{}^{C_1}Y}{{}^{C_1}Z} \qquad \gamma = \frac{1}{{}^{C_1}Z} \tag{3.26} $$

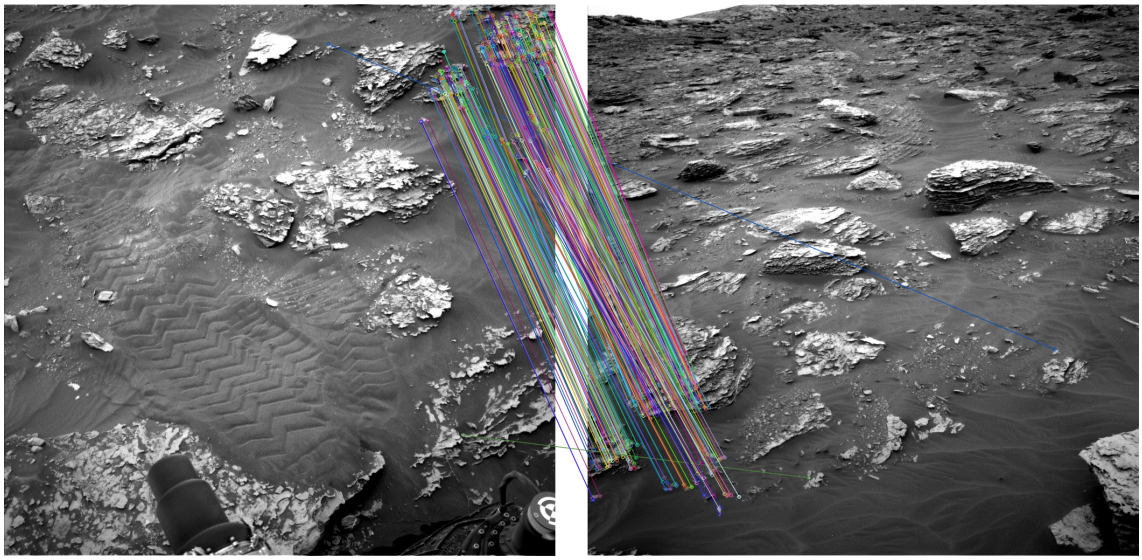


Figure 3.9: The result of feature matching from figure 3.6 on page 20. Note that there are at least two false positive matches, which even after close inspection look almost indistinguishable. The algorithm processing such data must be prepared for this situation and use some sort of outlier rejection.

The difference is that the parameter being optimized is $\boldsymbol{\theta}$. On close inspection, it can be seen that the parameters α and β define the ray on which the feature lays and γ is the inverse of the feature depth. As the depth of the feature approaches infinity, the parameter γ simply goes towards the zero and parameters α and β converge smoothly towards the correct value.

The error function can then be reformulated using the function $\mathbf{g}_i(\boldsymbol{\theta})$.

$$E(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n} \left\|\mathbf{z}_i - \mathbf{h}\left(\mathbf{g}_i(\boldsymbol{\theta})\right)\right\|^2 \tag{3.27}$$

This function is then minimized by using Gauss-Newton minimization or any other optimization technique.

When the minimal $\boldsymbol{\theta}$ is found, it can be used to estimate ${}^{G}\mathbf{p}_f$ such that

$${}^{G}\mathbf{p}_f = \frac{1}{\gamma}\,{}^{G}_{C_1}\mathbf{R}\begin{bmatrix}\alpha\\ \beta\\ 1\end{bmatrix} + {}^{G}\mathbf{p}_{C_1} \tag{3.28}$$

3.4 Kalman filtering

3.4.1 Kalman filter

Kalman filter is a commonly used algorithm. If the assumptions are met, it can optimally estimate the state of a system by utilizing a series of noisy measurements. More on this topic can be found in the book *Probabilistic Robotics* [1].

The schema of the Kalman filter is as follows

Algorithm 1 Kalman filter

Initial estimate

$$\mathbf{x}_0 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{0|0}, \boldsymbol{\Sigma}_{0|0}\right)$$

Propagation step

$$\begin{aligned}
\textbf{Given: } \mathbf{x}_k &= \mathbf{A}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_k\right) \\
\hat{\mathbf{x}}_{k|k-1} &= \mathbf{A}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k \\
\boldsymbol{\Sigma}_{k|k-1} &= \mathbf{A}_k\boldsymbol{\Sigma}_{k-1|k-1}\mathbf{A}_k^T + \mathbf{Q}_k
\end{aligned}$$

Update step

$$\begin{aligned}
\textbf{Given: } \mathbf{z}_k &= \mathbf{C}_k\mathbf{x}_k + \boldsymbol{v}_k, \quad \boldsymbol{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right) \\
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{C}_k\hat{\mathbf{x}}_{k|k-1}\right) \\
\boldsymbol{\Sigma}_{k|k} &= \boldsymbol{\Sigma}_{k|k-1} - \mathbf{K}_k\mathbf{C}_k\boldsymbol{\Sigma}_{k|k-1} \\
\mathbf{K}_k &= \boldsymbol{\Sigma}_{k|k-1}\mathbf{C}_k^T\left(\mathbf{C}_k\boldsymbol{\Sigma}_{k|k-1}\mathbf{C}_k^T + \mathbf{R}_k\right)^{-1}
\end{aligned}$$

where the variables are

$\mathbf{x}_k$	filter state at time k
$\boldsymbol{\Sigma}_k$	covariance matrix of the filter state at time k
$\mathbf{A}_k$	state-transition matrix
$\mathbf{B}_k$	control-input model
$\mathbf{C}_k$	measurement model
$\mathbf{K}_k$	Kalman gain

The propagation step is also known as prediction step.

3.4.2 Extended Kalman filter

When the state transition function and measurement functions are not linear, the standard Kalman filter cannot be used directly. Instead, both functions need to be linearized first. Such a filter is no longer optimal in general case (if the function is linear, EKF becomes a standard Kalman filter which is optimal).

It is easy to see that the extended Kalman filter is in fact very similar to the standard Kalman filter. Matrices $\mathbf{A}_k$ and $\mathbf{B}_k$ are replaced by a general function $\mathbf{f}(\cdot)$ and matrix $\mathbf{C}_k$ is replaced by function $\mathbf{h}(\cdot)$.

Algorithm 2 Extended Kalman filter

Initial estimate

$$\mathbf{x}_0 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{0|0}, \boldsymbol{\Sigma}_{0|0}\right)$$

Propagation step

$$\begin{aligned}
\textbf{Given: } \mathbf{x}_k &= \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_k\right) \\
\hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right) \\
\mathbf{F}_k &= \frac{\partial\, \mathbf{f}\left(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right)}{\partial\, \hat{\mathbf{x}}_{k-1|k-1}} \\
\boldsymbol{\Sigma}_{k|k-1} &= \mathbf{F}_k \boldsymbol{\Sigma}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k
\end{aligned}$$

Update step

$$\begin{aligned}
\textbf{Given: } \mathbf{z}_k &= \mathbf{h}\left(\mathbf{x}_k\right) + \boldsymbol{v}_k, \quad \boldsymbol{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right) \\
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{h}\left(\hat{\mathbf{x}}_{k|k-1}\right)\right) \\
\mathbf{H}_k &= \frac{\partial\, \mathbf{h}\left(\hat{\mathbf{x}}_{k|k-1}\right)}{\partial\, \hat{\mathbf{x}}_{k|k-1}} \\
\boldsymbol{\Sigma}_{k|k} &= \boldsymbol{\Sigma}_{k|k-1} - \mathbf{K}_k \mathbf{H}_k \boldsymbol{\Sigma}_{k|k-1} \\
\mathbf{K}_k &= \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^T \left(\mathbf{H}_k \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}
\end{aligned}$$

3.4.3 Indirect Kalman Filter

For some applications, most notably the Inertial Navigation System (INS), the functions involved in EKF suffer from significant nonlinearities, making it difficult for the filter to converge. One possible solution to this problem is to use an indirect Kalman filter. It is not a different type of a Kalman filter. In fact, it is a form of EKF where the problem is reformulated to improve the linearization properties.

The state $\mathbf{x}_k$ of the original EKF filter is split into a *nominal-state* $\hat{\mathbf{x}}_k$[6] and an *error-state* $\tilde{\mathbf{x}}_k$. $\mathbf{x}_k$ is then called a *total-state*. For these it holds that

$$\mathbf{x}_k = \hat{\mathbf{x}}_k \oplus \tilde{\mathbf{x}}_k \tag{3.29}$$

where $\oplus$ is a general composition operator, appropriate for the particular application. It is often just a standard addition.

In the EKF, the state-transition function was

$$\mathbf{x}_k = \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k \tag{3.30}$$

[6]The operator $\hat{\cdot}$ is overloaded. In the previous examples, it denoted the estimate of a variable, whereas in the indirect Kalman filter it denotes the nominal-state. This convention is widely used in the literature and despite the confusion, this work conforms to it. In most places, the meaning of the symbol can easily be inferred from the context.

By combining the last two equations, it can be seen that

$$\mathbf{x}_k = \mathbf{f}\left(\hat{\mathbf{x}}_k \oplus \tilde{\mathbf{x}}_k, \mathbf{u}_k\right) + \mathbf{w}_k \tag{3.31}$$

$$= \hat{\mathbf{f}}\left(\hat{\mathbf{x}}_k, \mathbf{u}_k\right) \oplus \tilde{\mathbf{f}}\left(\tilde{\mathbf{x}}_k, \mathbf{u}_k\right) + \mathbf{w}_k \tag{3.32}$$

with $\hat{\mathbf{f}}$ being the nominal-state transition function and $\tilde{\mathbf{f}}$ being the error-state transition function.

The function $\hat{\mathbf{f}}$ is chosen so that it consists of all the non-noise-related terms of the original function $\mathbf{f}$. The noise-related terms, which are not observable during the propagation step and need to be corrected by the update step, are part of the function $\tilde{\mathbf{f}}$. Because of that, the mean value of the error-state estimate does not change during the propagation an therefore it holds that

$$\tilde{\mathbf{x}}_{k|k-1} = \tilde{\mathbf{f}}\left(\tilde{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\right) \tag{3.33}$$

$$= \tilde{\mathbf{x}}_{k-1|k-1} \tag{3.34}$$

To reflect the change in noise parameters, the uncertainty of the error-state, expressed by the covariance matrix $\mathbf{\Sigma}_k$, must necessarily grow during the propagation. The nominal-state has no uncertainty as it was defined to only contain the known terms.

During the update step of the EKF, the measurement function is

$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \mathbf{v}_k \tag{3.35}$$

To linearize the function $\mathbf{h}\left(\cdot\right)$, it is approximated by the first two terms of a Taylor series, expanded at point $\hat{\mathbf{x}}_k$ which is the best estimate of the total state. This leads to

$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \mathbf{v}_k \tag{3.36}$$

$$= \mathbf{h}\left(\hat{\mathbf{x}}_k\right) + \frac{\partial\, \mathbf{h}\left(\hat{\mathbf{x}}_k\right)}{\partial\, \tilde{\mathbf{x}}_k}\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right) + \mathbf{v}_k \tag{3.37}$$

$$\mathbf{z}_k - \hat{\mathbf{z}}_k = \frac{\partial\, \mathbf{h}\left(\hat{\mathbf{x}}_k\right)}{\partial\, \tilde{\mathbf{x}}_k}\tilde{\mathbf{x}}_k + \mathbf{v}_k \tag{3.38}$$

$$\mathbf{r}_k = \mathbf{H}_k\tilde{\mathbf{x}}_k + \mathbf{v}_k \tag{3.39}$$

After examining the equation, it can be seen that it is in the form of the measurement function of the Kalman filter. It means that this equation serves as a proper measurement model. The measurement entering the indirect Kalman filter is the residual vector $\mathbf{r}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k$ and the measurement model is the matrix $\mathbf{H}_k$.

After the update step is performed, the error state is estimated. Its mean value is then transferred into the nominal state, which does not change the value of the total-state. The covariance matrix of the error-state needs to be adjusted accordingly. This effectively resets the error-state to zero after every update (the covariance matrix is not reset to zero). This step is also called a reset procedure.

The structure of the indirect Kalman filter is summarized in the algorithm below. Operator $\ominus$ is the counterpart of the operator $\oplus$ mentioned earlier.

Algorithm 3 Indirect Kalman filter

Initial estimate

$$\mathbf{x}_0 \sim \mathcal{N}\left(\hat{\mathbf{x}}_{0|0}, \boldsymbol{\Sigma}_{0|0}\right)$$

Propagation step

$$\textbf{Given: } \mathbf{x}_k = \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_k\right)$$

nominal-state	**error-state**
$\hat{\mathbf{x}}_{k\|k-1} = \hat{\mathbf{f}}\left(\hat{\mathbf{x}}_{k-1\|k-1}, \mathbf{u}_k\right)$	$\mathbf{F}_k = \dfrac{\partial\, \tilde{\mathbf{f}}\left(\tilde{\mathbf{x}}_{k-1\|k-1}, \mathbf{u}_k\right)}{\partial\, \tilde{\mathbf{x}}_{k-1\|k-1}}$
	$\boldsymbol{\Sigma}_{k\|k-1} = \mathbf{F}_k \boldsymbol{\Sigma}_{k-1\|k-1} \mathbf{F}_k^T + \mathbf{Q}_k$

Update step

$$\textbf{Given: } \mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \boldsymbol{v}_k, \quad \boldsymbol{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right)$$

nominal-state	**error-state**
$\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1}$	$\hat{\tilde{\mathbf{x}}}_{k\|k} = \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{h}\left(\hat{\mathbf{x}}_{k\|k-1}\right)\right)$
	$\mathbf{H}_k = \dfrac{\partial\, \mathbf{h}\left(\hat{\mathbf{x}}_{k\|k-1}\right)}{\partial\, \tilde{\mathbf{x}}_{k\|k-1}}$
	$\boldsymbol{\Sigma}_{k\|k} = \boldsymbol{\Sigma}_{k\|k-1} - \mathbf{K}_k \mathbf{H}_k \boldsymbol{\Sigma}_{k\|k-1}$
	$\mathbf{K}_k = \boldsymbol{\Sigma}_{k\|k-1} \mathbf{H}_k^T \left(\mathbf{H}_k \boldsymbol{\Sigma}_{k\|k-1} \mathbf{H}_k^T + \mathbf{R}_k\right)^{-1}$

Update step: Injection

nominal-state	**error-state**
$\hat{\mathbf{x}}_{k\|k} \leftarrow \hat{\mathbf{x}}_{k\|k} \oplus \hat{\tilde{\mathbf{x}}}_{k\|k}$	$\boldsymbol{\Sigma}_{k\|k} \leftarrow \mathbf{G}_k \boldsymbol{\Sigma}_{k\|k} \mathbf{G}_k^T$
	$\mathbf{G}_k = \dfrac{\partial\, \left(\tilde{\mathbf{x}}_{k\|k} \ominus \hat{\tilde{\mathbf{x}}}_{k\|k}\right)}{\partial\, \tilde{\mathbf{x}}_{k\|k}}$
	$\hat{\tilde{\mathbf{x}}}_{k\|k} \leftarrow \mathbf{0}$

The advantages of *indirect Kalman Filter* for visual-inertial odometry (VIO) (and INS in general) are twofold. First, as mentioned above, the *true-state* dynamics often exhibit large nonlinearities which effectively limits the estimation accuracy of the EKF, whereas the nonlinearity of the well-designed error-state transition function should be better [49]. Second, Nyquist–Shannon sampling theorem implies limitations on the update rate of the filter with *total-state* representation. It would need to be at least twice as high as the highest frequency of the signal being estimated (although according to Maybeck [49] 5x to 10x is usually used in practice). This would require sensors capable of sampling at these frequencies and it would also induce a high computational load. Because the error-state evolves relatively slowly in time, much lower frequency of the update

steps is needed. For some systems, the update period might be 10s of seconds instead of the IMU-rate (which might be 100 Hz - 10 kHz).

4. Multi-State Constraint Kalman Filter

MSCKF is a form of *indirect Kalman Filter* used for accurate estimation of robot's pose in 6 degrees of freedom (3 for position and 3 for orientation in 3D). It fuses an IMU and a camera to achieve real-time performance with low position estimate drift.

The key point of the MSCKF is its measurement model (used in the update step of the Kalman filter), that is able to express the geometric constraints that arise when a static feature is observed from multiple camera poses. It does not require including the 3D feature position in the state vector of the EKF. Instead of that, it keeps a rolling window of the last few camera poses in the state. "This procedure is optimal up to the linearization errors" [19].

4.1 Development of MSCKF

The MSCKF started as a relatively straightforward extension of the EFK-SLAM (discussed in section 1.2.1). Over the years, the MSCKF gained a lot of traction in terms of academic research. The following paragraphs briefly sketch the historical development as there are many different subtle changes that differ across many articles and are important in order to properly understand the inner workings.

The first article introducing the MSCKF was written by Mourikis and Roumeliotis in 2007, called *A Multi-state Constraint Kalman Filter for Vision-Aided Inertial Navigation* [19]. A precursor of this work is a technical report of the same name from the same authors. It was released a year earlier, in 2006, and describes the same filter in more detail [20].

It has been shown that the EKF-SLAM is an inconsistent algorithm [50, 51]. In principle, the position and rotation around gravity (yaw) should both be unobservable by the algorithm (this holds for any visual-inertial odometry). When implemented in a straight-forward way, the EKF-SLAM spuriously gains observability of the yaw. This inconsistency leads to underestimation of uncertainties and consequently to loss of the estimation accuracy to some degree. This problem is not only related to EKF-SLAM but also to other similar filtering-based techniques, including the MSCKF.

In 2012 Li and Mourikis [52] provided a thorough theoretical analysis of MSCKF, finding the root causes of the problem. They proposed to address the issue by expressing the angular errors in the global frame instead of the body frame and by using the *first-estimate jacobians*. This is one of the techniques previously used in EKF-SLAM to address the observability issue. The result was supported by both simulation and real-world experiments. They concluded that the estimation accuracy had been slightly improved and the consistency was improved dramatically.

Together with their article, a technical report was released [16]. It featured more detailed derivation of the results presented in the main paper. It is a good resource for understanding the filter.

Later in 2012, Li and Mourikis [53, 54] presented a fusion of the MSCKF and

EKF-SLAM. They proposed to keep the long-tracked features directly in the filter state (as EKF-SLAM does for all the features) and process the rest of the features in the MSCKF fashion. By carefully adjusting the size of the sliding window in runtime, they were able to achieve a real-time performance on a mobile phone (Samsung Galaxy S2 with Dual-core 1.2 GHz Cortex-A9 CPU and 1 GB RAM).

The visual-inertial algorithms are usually very sensitive to sensor synchronization. In research projects, it is usually solved by hardware synchronization. That is by no means available on mobile devices. This issue was addressed by Li and Mourikis in 2013 [55]. They let the filter to estimate both spatial and temporal relationship of the camera and the IMU. By theoretical analysis, they proved [56] that these properties are properly observable by the filter.

At about the same time, Li and Mourikis [23] modeled a rolling-shutter camera. They did so by only assuming a constant velocity during the time the image was being read. Other works by different authors typically used much stronger assumptions, leading to unmodeled errors.

The same authors continued with the trend of modeling the errors of lower-grade hardware. In 2014, they featured a version of MSCKF [24] capable of modeling and estimating many different sources of errors connected with this kind of hardware.

Other MSCKF extensions involve stereo cameras by Sun et al. [57], direct photometric error formulation by Zheng et al. [58], edges-based measurement model by Yu and Mourikis [59] and many others.

4.2 Algorithm overview

The filter uses IMU measurements to predict (propagate) the filter state. When an image arrives from the camera, the visual features are tracked using for example KLT tracker. When a feature is not detected in the latest image, its whole feature track is used to form a constraint on the filter state, correcting the accumulated errors.

The algorithm performs an online autocalibration of multiple parameters such as IMU biases, camera-to-IMU spatial and temporal transformation and others. This simplifies the real-world deployment because only a rough initial estimate of the parameters is needed, reducing the calibration process.

Because the time of the next image from camera cannot be predicted, all IMU measurements are buffered. When the image arrives, the buffered IMU measurements are used to propagate the filter state to the estimated time of the image capture, a new camera pose is appended to the rolling-window of camera poses kept in the state and the update step is performed.

The algorithm described in this chapter is based on many of the articles on the topic but is not an exact version of any of them. There are small variations in what quantities are being estimated, different measurement models for both IMU and camera, etc.

4.3 State description

4.3.1 Coordinate frames

There are three major coordinate frames used by the filter which also estimates their relative transformations.

The body frame $\{B\}$ is located at the position of the IMU and it is oriented in the same direction.

The origin of the camera frame is at the focal point of the camera. It is oriented as depicted in fig. (3.5b). That is, the y-axis points down in the image, x-axis to the right and z-axis point from the focal point of the camera towards the captured scene.

Both body and camera frame are fixed to the robot and move with it. The third, global frame is fixed to the world and does not move. The current robot pose is expressed in this frame. This frame is chosen such that when the filter starts, the position is zero vector (this is merely a convention and can be chosen arbitrarily, changing the meaning of the global frame).

4.3.2 Sensor timing

When using the camera and IMU measurements, they need to be labeled by the timestamp they were captured at. In research robots, this is commonly ensured by the hardware itself. The IMU emits a measurement, which triggers the camera and both samples are assigned a correct timestamp.

For commodity hardware, this is not the case. A common realization is that both camera and IMU are configured to emit measurements at a certain frequency and they start generating the measurements at about that rate asynchronously. As the sensors themselves usually do not have a precise hardware clock, a timestamp is assigned by the operating system at the time the measurement arrives. The actual delay is different for both sensors and can lead to tracking failures if not modeled properly. The precise delay cannot be estimated for neither of the sensors but it is possible and sufficient to estimate the relative difference between the two delays. The synchronization of the two streams can be achieved by subtracting the relative difference from the timestamps of one of them.

Another problem can be caused by the time between measurement capture and delivery being nondeterministic. That can be caused by any buffer before the timestamp assignment (the bus via which the sensor is connected, the OS scheduling the processing daemon to run or similar). This can be compensated for by treating the time offset as a random variable [25].

When the timestamps of each sensor are assigned by a different clock, a clock skew might occur. This happens when one of the clocks is slightly faster. The MSCKF can properly account for that as well.

4.3.3 State vector

The evolving IMU state is described by the vector

$$\mathbf{X}_{\text{IMU}} = \begin{bmatrix} {}^{B}_{G}\bar{\mathbf{q}}^T & {}^{G}\mathbf{p}_B^T & {}^{G}\mathbf{v}_B^T & \mathbf{b}_g^T & \mathbf{b}_a^T \end{bmatrix}^T \tag{4.1}$$

where ${}^B_G\bar{\mathbf{q}} \in \mathbb{R}^4$ is a unit quaternion representing the rotation from the global frame $\{G\}$ to the body frame $\{B\}$, ${}^G\mathbf{p}_B, {}^G\mathbf{v}_B \in \mathbb{R}^3$ are the position and velocity of the body frame in the global frame and $\mathbf{b}_g$ and $\mathbf{b}_a$, both from $\mathbb{R}^3$, are the biases of the gyroscope and accelerometer respectively, as in equations (3.1) and (3.4).

The IMU error-state $\tilde{\mathbf{X}}_{\text{IMU}}$ is defined as follows

$$\tilde{\mathbf{X}}_{\text{IMU}} = \begin{bmatrix} {}^B_G\tilde{\boldsymbol{\theta}}^T & {}^G\tilde{\mathbf{p}}_B^T & {}^G\tilde{\mathbf{v}}_B^T & \tilde{\mathbf{b}}_g^T & \tilde{\mathbf{b}}_a^T \end{bmatrix}^T \tag{4.2}$$

All the quantities but rotation error have the same dimension as their corresponding quantities from the nominal-state and a standard additive error is used (e.g. ${}^G\mathbf{p}_B = {}^G\hat{\mathbf{p}}_B + {}^G\tilde{\mathbf{p}}_B$). For the quaternion, a different operation has to be used. Let $\bar{\mathbf{q}}$ be the true-state orientation and $\hat{\bar{\mathbf{q}}}$ be the nominal-state one, then the orientation error is expressed as an error quaternion $\tilde{\bar{\boldsymbol{q}}}$ satisfying $\bar{\mathbf{q}} = \tilde{\bar{\boldsymbol{q}}} \otimes \hat{\bar{\mathbf{q}}}$, where $\otimes$ is quaternion multiplication. $\tilde{\bar{\boldsymbol{q}}}$ represents a small rotation of angle θ around an axis $\mathbf{k}$ and can be rewritten by eq. (A.17) as

$$\tilde{\bar{\boldsymbol{q}}} \simeq \begin{bmatrix} \frac{1}{2}\tilde{\boldsymbol{\theta}}^T & 1 \end{bmatrix}^T \tag{4.3}$$

Such simplification has an advantage of being the minimal representation of a rotation in 3D. As a consequence, we do not need to enforce the unit-length of the quaternions during the update step or similar conditions for alternative representations of the rotations.

The filter state has variable number of body poses $\boldsymbol{\pi}_{B_i}$. The body pose estimate at time t_c, which is the estimated time when the corresponding image was captured, is

$$\boldsymbol{\pi}_{B_i} = \begin{bmatrix} {}^{B_i}_G\bar{\mathbf{q}}^T & {}^G\mathbf{p}_{B_i}^T & {}^G\mathbf{v}_{B_i}^T \end{bmatrix}^T \tag{4.4}$$

The subscript k denotes the filter step performed at time t_k. The next step is $k+1$ at time t_{k+1} and $\Delta t = t_{k+1} - t_k$. In general, filter steps do not need to be evenly spaced in time.

At a given time step k, the state vector $\mathbf{X}_k \in \mathbb{R}^{24+7N}$ contains N body poses and is

$$\mathbf{X}_k = \begin{bmatrix} \mathbf{X}_{\text{IMU}}^T & {}^B_C\bar{\mathbf{q}}^T & {}^B\mathbf{p}_C^T & t_d & \boldsymbol{\pi}_{B_0}^T & \dots & \boldsymbol{\pi}_{B_{N-1}}^T \end{bmatrix}^T \tag{4.5}$$

with ${}^B_C\bar{\mathbf{q}}^T$ being a unit quaternion representing the rotation from camera to body frame, ${}^B\mathbf{p}_C^T$ being the position of the camera frame in the body frame. t_d is the time difference between the camera and IMU clocks. When an image is delivered with a timestamp of t, its estimated capture time is $t_c = t + t_d$. The number of camera poses N is variable in time but has an upper bound of $N_{\max}$.

Finally, the error-state $\tilde{\mathbf{X}}_k \in \mathbb{R}^{22+6N}$ is

$$\tilde{\mathbf{X}}_k = \begin{bmatrix} \tilde{\mathbf{X}}_{\text{IMU}}^T & {}^B_C\tilde{\boldsymbol{\theta}}^T & {}^B\tilde{\mathbf{p}}_C^T & \tilde{t}_d & \tilde{\boldsymbol{\pi}}_{B_0}^T & \dots & \tilde{\boldsymbol{\pi}}_{B_{N-1}}^T \end{bmatrix}^T \tag{4.6}$$

where ${}^B_C\tilde{\boldsymbol{\theta}} \in \mathbb{R}^3$ and ${}^B\tilde{\mathbf{p}}_C \in \mathbb{R}^3$ are the orientation and position error of the camera-to-IMU transformation and $\tilde{t}_d$ is the error of the camera-to-IMU time offset. The camera pose part of the error-state is $\tilde{\boldsymbol{\pi}}_{B_i} \in \mathbb{R}^6$ such that

$$\tilde{\boldsymbol{\pi}}_{B_i} = \begin{bmatrix} {}^{B_i}_G\tilde{\boldsymbol{\theta}}^T & {}^G\tilde{\mathbf{p}}_{B_i}^T & {}^G\tilde{\mathbf{v}}_{B_i}^T \end{bmatrix}^T \tag{4.7}$$

4.4 Propagation

In order to explain the propagation step, the system behavior (dynamics) in continuous-time is first described for both nominal- and error-state. The nominal-state dynamics are used to propagate the nominal-state and the error-state dynamic are used to propagate the covariance matrix.

The true- IMU-state dynamics can be modeled in continuous-time using the following equations

$$ {}^{B}_{G}\dot{\bar{\mathbf{q}}}(t) = \tfrac{1}{2}\boldsymbol{\Omega}\left({}^{B}\boldsymbol{\omega}(t)\right){}^{B}_{G}\bar{\mathbf{q}}(t) \tag{4.8} $$

$$ \dot{\mathbf{b}}_g(t) = \mathbf{n}_{wg}(t) \tag{4.9} $$

$$ {}^{G}\dot{\mathbf{v}}_B(t) = {}^{G}\mathbf{a}(t) \tag{4.10} $$

$$ \dot{\mathbf{b}}_a(t) = \mathbf{n}_{wa}(t) \tag{4.11} $$

$$ {}^{G}\dot{\mathbf{p}}_B(t) = {}^{G}\mathbf{v}_B(t) \tag{4.12} $$

where ${}^{B}\boldsymbol{\omega}(t) = \begin{bmatrix}\omega_x & \omega_y & \omega_z\end{bmatrix}^T$ is the rotation rate measured in $\{B\}$, ${}^{G}\mathbf{a}(t)$ is the acceleration measured in $\{G\}$, and

$$ \boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -\lfloor\boldsymbol{\omega}_\times\rfloor & \boldsymbol{\omega} \\ \boldsymbol{\omega}^T & 0 \end{bmatrix}, \qquad \lfloor\boldsymbol{\omega}_\times\rfloor = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} $$

Note that $\lfloor\boldsymbol{\omega}_\times\rfloor$ is antisymmetric (skew-symmetric) matrix and it represents a matrix notation of the vector cross product - for two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ holds that $\mathbf{a} \times \mathbf{b} = \lfloor \boldsymbol{a}_\times \rfloor \mathbf{b}$. $\mathbf{n}_{wg}$ from eq. (4.9) and $\mathbf{n}_{wa}$ from eq. (4.11) are the random walk parameters as described in sections 3.2.1 and 3.2.2 respectively.

From the eqs. (3.1, 3.4), the estimated acceleration and rotation rate are

$$ {}^{B}\hat{\boldsymbol{\omega}}(t) = {}^{B}\boldsymbol{\omega}_m(t) - \mathbf{b}_g \tag{4.13} $$

$$ {}^{B}\hat{\mathbf{a}}(t) = {}^{B}\mathbf{a}_m(t) - \mathbf{b}_a \tag{4.14} $$

The error- IMU-state dynamics in continuous-time is

$$ {}^{B}_{G}\dot{\tilde{\boldsymbol{\theta}}}(t) = -\left\lfloor {}^{B}\hat{\boldsymbol{\omega}}(t)_\times \right\rfloor {}^{B}_{G}\tilde{\boldsymbol{\theta}}(t) - \tilde{\boldsymbol{b}}_g(t) - \boldsymbol{n}_g \tag{4.15} $$

$$ \dot{\tilde{\mathbf{b}}}_g(t) = \mathbf{n}_{wg} \tag{4.16} $$

$$ {}^{G}\dot{\tilde{\mathbf{v}}}_B(t) = -{}^{B}_{G}\mathbf{R}(t)^T \left\lfloor {}^{B}\hat{\boldsymbol{a}}(t)_\times \right\rfloor {}^{B}_{G}\tilde{\boldsymbol{\theta}}(t) - {}^{B}_{G}\mathbf{R}(t)^T\tilde{\boldsymbol{b}}_a(t) - {}^{B}_{G}\mathbf{R}^T(t)\,\mathbf{n}_a \tag{4.17} $$

$$ \dot{\tilde{\mathbf{b}}}_a(t) = \mathbf{n}_{wa} \tag{4.18} $$

$$ {}^{G}\dot{\tilde{\mathbf{p}}}_B(t) = {}^{G}\tilde{\boldsymbol{v}}_B(t) \tag{4.19} $$

where the $\boldsymbol{n}_g$ a $\boldsymbol{n}_a$ are Gaussian noises of each measurements, described in sections 3.2.1 and 3.2.2.

Derivation of equations (4.8 - 4.12) can be found in a text by Solà [60]. Matrix ${}^{B}_{G}\mathbf{R}(t)$ is obtained by converting the corresponding quaternion ${}^{B}_{G}\mathbf{q}(t)$ from the nominal-state, as described in eq. (A.6). Linearizing the error- IMU-state results in

$$ \dot{\tilde{\boldsymbol{X}}}_{\text{IMU}} = \boldsymbol{F}\tilde{\boldsymbol{x}}_{\text{IMU}} + \boldsymbol{G}\boldsymbol{n}_{\text{IMU}} \tag{4.20} $$

where $\boldsymbol{n}_{\text{IMU}}$ is a vector of noises influencing the IMU state defined as

$$\boldsymbol{n}_{\text{IMU}} = \begin{bmatrix} \boldsymbol{n}_g^T & \boldsymbol{n}_{wg}^T & \boldsymbol{n}_a^T & \boldsymbol{n}_{wa}^T \end{bmatrix}^T \tag{4.21}$$

Using the equations (4.15) to (4.19), matrices $\boldsymbol{F}$ and $\boldsymbol{G}$ from equation (4.20) are

$$\boldsymbol{F} = \begin{bmatrix} -\left\lfloor {}^B\hat{\boldsymbol{\omega}}(t)_\times \right\rfloor & \mathbf{0}_3 & \mathbf{0}_3 & -\mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ -{}^B_G\mathbf{R}(t)^T \left\lfloor {}^B\hat{\mathbf{a}}(t)_\times \right\rfloor & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & -{}^B_G\mathbf{R}(t)^T \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix} \tag{4.22}$$

$$\boldsymbol{G} = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & -{}^B_G\mathbf{R}(t)^T & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix} \tag{4.23}$$

4.4.1 Orientation propagation

As mentioned in section 3.2.1, the gyroscope measurements ${}^B\boldsymbol{\omega}_m(t)$ are corrupted by various factors. The *gyroscope measurement model* used in this work (described in section 3.2.2) takes into account the bias $\mathbf{b}_a$ and a zero-mean, white Gaussian noise $\mathbf{n}_a$

$${}^B\hat{\boldsymbol{\omega}}_m(t) = {}^B\hat{\boldsymbol{\omega}}(t) + \mathbf{b}_a + \mathbf{n}_a \tag{4.24}$$

Because $\mathbf{n}_a$ has zero mean, it is removed from the propagation equations but it still influences the covariance of the propagation step.

The goal of orientation propagation is to find the quaternion ${}^{B_{k+1}}_{B_k}\hat{\mathbf{q}}$, rotating the body frame $\{B_k\}$ to $\{B_{k+1}\}$ and satisfying

$${}^{B_{k+1}}_{G}\hat{\mathbf{q}} = {}^{B_{k+1}}_{B_k}\hat{\mathbf{q}} \otimes {}^{B_k}_{G}\hat{\mathbf{q}} \tag{4.25}$$

It can be obtained by numerically integrating the equation (4.8) on the interval $[t_k, t_{k+1}]$, starting from the unit quaternion. By using the fourth order Runge-Kutta integration schema, it is

$${}^{B_{k+1}}_{B_k}\hat{\mathbf{q}} = \bar{\mathbf{q}}_0 + \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{4.26}$$

$$\bar{\mathbf{q}}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{4.27}$$

$$\mathbf{k}_1 = \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}(t_k)\right)\bar{\mathbf{q}}_0 \tag{4.28}$$

$$\mathbf{k}_2 = \frac{1}{2}\boldsymbol{\Omega}\left(\frac{\hat{\boldsymbol{\omega}}(t_k) + \hat{\boldsymbol{\omega}}(t_{k+1})}{2}\right)\left(\bar{\mathbf{q}}_0 + \frac{\Delta t}{2}\mathbf{k}_1\right) \tag{4.29}$$

$$\mathbf{k}_3 = \frac{1}{2}\boldsymbol{\Omega}\left(\frac{\hat{\boldsymbol{\omega}}(t_k) + \hat{\boldsymbol{\omega}}(t_{k+1})}{2}\right)\left(\bar{\mathbf{q}}_0 + \frac{\Delta t}{2}\mathbf{k}_2\right) \tag{4.30}$$

$$\mathbf{k}_4 = \frac{1}{2}\boldsymbol{\Omega}\left(\hat{\boldsymbol{\omega}}(t_{k+1})\right)\left(\bar{\mathbf{q}}_0 + \Delta t\mathbf{k}_3\right) \tag{4.31}$$

Because this numerical integration does not preserve the length of a vector, it needs to be normalized to a unit vector in order to represent a valid rotation quaternion

$$\hat{\mathbf{q}}_{B_k}^{B_{k+1}} \leftarrow \frac{\hat{\mathbf{q}}_{B_k}^{B_{k+1}}}{\left\|\hat{\mathbf{q}}_{B_k}^{B_{k+1}}\right\|} \tag{4.32}$$

4.4.2 Position and velocity propagation

To derive the position and velocity propagation, an *accelerometer measurement model* is needed. Similarly to the gyroscope measurement model, the model used in this work takes into account the accelerometer bias $\mathbf{b}_a$ and a zero-mean, white Gaussian noise $\mathbf{n}_a$ resulting in

$${}^{B}\mathbf{a}_m = {}^{B}_{G}\mathbf{R}\left({}^{G}\mathbf{a} - {}^{G}\mathbf{g}\right) + \mathbf{b}_a + \mathbf{n}_a \tag{4.33}$$

where

$${}^{G}\mathbf{g} = \begin{bmatrix} 0 & 0 & -9.8xx \end{bmatrix}^T \tag{4.34}$$

The "xx" in $-9.8xx$ should be replaced with the appropriate value of the gravitational acceleration at a given location.

The velocity propagation can be derived by integrating differential equation (4.10) on the interval $[t_k, t_{k+1}]$ with the initial condition of ${}^{G}\hat{\mathbf{v}}_{t_k}$.

$$\begin{aligned}
{}^{G}\hat{\mathbf{v}}_{k+1} &= {}^{G}\hat{\mathbf{v}}_k + \int_{t_k}^{t_{k+1}} {}^{G}\hat{\mathbf{a}}(\tau)\, d\tau && (4.35)\\
&= {}^{G}\hat{\mathbf{v}}_k + \int_{t_k}^{t_{k+1}} \left({}^{G}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau) + {}^{G}\mathbf{g}\right) d\tau && (4.36)\\
&= {}^{G}\hat{\mathbf{v}}_k + {}^{G}_{B_k}\hat{\mathbf{R}} \int_{t_k}^{t_{k+1}} {}^{B_k}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau) d\tau + {}^{G}\mathbf{g}\Delta t && (4.37)\\
&= {}^{G}\hat{\mathbf{v}}_k + {}^{G}_{B_k}\hat{\mathbf{R}}\hat{\mathbf{s}}_k + {}^{G}\mathbf{g}\Delta t && (4.38)
\end{aligned}$$

The position can be obtained in a similar way by integrating equation (4.12) on the same interval, starting at the current estimate

$$\begin{aligned}
{}^{G}\hat{\mathbf{p}}_{k+1} &= {}^{G}\hat{\mathbf{p}}_k + \int_{t_k}^{t_{k+1}} {}^{G}\hat{\mathbf{v}}(\tau)\, d\tau && (4.39)\\
&= {}^{G}\hat{\mathbf{p}}_k + {}^{G}\hat{\mathbf{v}}_k\Delta t + {}^{G}_{B_k}\hat{\mathbf{R}}\hat{\mathbf{y}}_k + \frac{1}{2}{}^{G}\mathbf{g}\Delta t^2 && (4.40)
\end{aligned}$$

where the vectors $\hat{\mathbf{s}}_k$ and $\hat{\mathbf{y}}_k$ are

$$\begin{aligned}
\hat{\mathbf{s}}_k &= \int_{t_k}^{t_{k+1}} {}^{B_k}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau && (4.41)\\
\hat{\mathbf{y}}_k &= \int_{t_k}^{t_{k+1}} \int_{t_k}^{s} {}^{B_k}_{B_\tau}\hat{\mathbf{R}}\,{}^{B}\hat{\mathbf{a}}(\tau)\, d\tau\, ds && (4.42)
\end{aligned}$$

The solution obtained from the *Euler integration*[1] is

$$\hat{\mathbf{s}}_k \simeq \frac{\Delta t}{2}\left({}^{B_k}_{B_{k+1}}\hat{\mathbf{R}}\left({}^{B}\mathbf{a}_m(t_{k+1}) - \hat{\mathbf{b}}_a\right)\right) \tag{4.43}$$

$$\hat{\mathbf{y}}_k \simeq \frac{\Delta t}{2}\hat{\mathbf{s}}_k \tag{4.44}$$

4.4.3 Error propagation

The covariance matrix of the error-state from time t_k for time t_{k+1} is $\mathbf{\Sigma}_{k+1|k}$. It can be obtained using the error-state transition matrix $\mathbf{\Phi}_k$ such that

$$\mathbf{\Sigma}_{k+1|k} = \mathbf{\Phi}_k \mathbf{\Sigma}_{k|k} \mathbf{\Phi}_k^T + \mathbf{Q}_d \tag{4.45}$$

where

$$\mathbf{\Phi}_k = \begin{bmatrix} \mathbf{\Phi}_{\mathrm{IMU}_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \qquad \mathbf{Q}_d = \begin{bmatrix} \mathbf{Q}_k & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{4.46}$$

From [17], it holds that[2]

$$\mathbf{\Phi}_{\mathrm{IMU}_k} = \mathbf{\Phi}(t_{k+1}, t_k) = \exp\left(\int_{t_k}^{t_{k+1}} \mathbf{F}(\tau)\, d\tau\right) \tag{4.47}$$

$$\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1}, \tau)\, \mathbf{G}\, \mathbf{Q}_c\, \mathbf{G}^T\, \mathbf{\Phi}(t_{k+1}, \tau)^T\, d\tau \tag{4.48}$$

$$\mathbf{Q}_c = \mathbf{Diag}\left(\begin{bmatrix} \sigma_g^2 \mathbb{1}_3^T & \sigma_{wg}^2 \mathbb{1}_3^T & \sigma_a^2 \mathbb{1}_3^T & \sigma_{wa}^2 \mathbb{1}_3^T \end{bmatrix}^T\right) \tag{4.49}$$

Values σ_g^2, σ_{wg}^2, σ_a^2, σ_{wa}^2 are IMU bias and noise characteristics. They can be either supplied by the IMU manufacturer in the device's datasheet or obtained from offline calibration [61].

4.5 State augmentation

When a new image from the camera arrives at time t, the filter state is propagated to time $t + t_d$ using the buffered IMU measurements, a new camera pose is appended to the filter state. The covariance matrix needs to be adjusted accordingly

$$\mathbf{\Sigma}_k \leftarrow \begin{bmatrix} \mathbf{I}_{22+6*N} \\ \mathbf{J}_{\boldsymbol{\pi}} \end{bmatrix} \mathbf{\Sigma}_k \begin{bmatrix} \mathbf{I}_{22+6*N} \\ \mathbf{J}_{\boldsymbol{\pi}} \end{bmatrix}^T \tag{4.50}$$

where $\mathbf{\Sigma}_k$ is the covariance matrix

$$\mathbf{J}_{\boldsymbol{\pi}} = \frac{\partial \tilde{\mathbf{X}}_{\mathbf{k}}}{\partial \tilde{\boldsymbol{\pi}}_B} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \dots \end{bmatrix} \tag{4.51}$$

[1] The Euler integration is

$$\int_a^b f(x)\, dx = \frac{b-a}{2}\left(f(a) + f(b)\right)$$

[2] Note that

$$\exp \mathbf{A} = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{A}^k$$

4.6 Update step

In order to draw the error-state observable, a fusion with a complementary sensor is necessary. In this work, a monocular camera is used.

Since the MSCKF is based on the EKF, some measurement model (here called $h_{\text{EKF}}(\cdot)$) needs to be defined such that the residual vector $\mathbf{r}$ depends linearly on the filter state - $\tilde{\mathbf{X}}$. The general form is

$$\mathbf{r} = \mathbf{H}\tilde{\mathbf{X}} + \text{noise} \tag{4.52}$$

By the assumptions of the EKF, the noise term has to be white and uncorrelated with $\tilde{\mathbf{X}}$. $\mathbf{H}$ is the Jacobian matrix of the measurement model with respect to the error-state, evaluated at the nominal-state

$$\mathbf{H} = \left.\frac{\partial h_{\text{EKF}}(\mathbf{x})}{\partial \tilde{\mathbf{X}}}\right|_{\mathbf{x}=\hat{\mathbf{X}}} \tag{4.53}$$

4.6.1 Single-feature measurement model

The measurement used in the update step consists of all the observations of the features that were tracked in the previous frames but are not tracked in the current one. For the sake of simplicity, we first describe a case with a single tracked feature.

A feature f_j was observed from a set of camera poses $\mathcal{S}_j, |\mathcal{S}_j| = M_j$. Each pose is described by the IMU pose $\boldsymbol{\pi}_{B_i} \in \mathcal{S}_j$ from the time the image was captured. The feature is projected onto the image plane by applying the *pinhole camera model* $h(\cdot)$, described in section 3.3.1

$$\mathbf{z}_i^{(j)} = h\left({}^{C_i}\mathbf{p}_{f_j}\right) = \frac{1}{{}^{C_i}Z_j}\begin{bmatrix} {}^{C_i}X_j \\ {}^{C_i}Y_j \end{bmatrix} + \mathbf{n}_i^{(j)} \tag{4.54}$$

where $\mathbf{z}_i^{(j)}$ is the 2D position of the feature projected onto the image plane, ${}^{C_i}\mathbf{p}_{f_j} = \begin{bmatrix} {}^{C_i}X_j & {}^{C_i}Y_j & {}^{C_i}Z_j \end{bmatrix}^T$ is the position of the jth feature in ith camera pose and $\mathbf{n}_i^{(j)} \in \mathbb{R}^{2\times 1}$ is the image noise vector with the covariance of $\sigma_{\text{im}}^2\mathbf{I}_2$.

The function ρ_i calculates the position of the feature in the ith camera pose, based on its position in the global frame and the state.

$${}^{C_i}\mathbf{p}_{f_j} = \rho_i\left(\mathbf{X}, {}^{G}\mathbf{p}_{f_j}\right) \tag{4.55}$$

$$\rho_i\left(\mathbf{X}, {}^{G}\mathbf{p}_{f_j}\right) = {}^{C}_{B}\mathbf{R}\,{}^{B_i}_{G}\mathbf{R}(t+t_d)\left({}^{G}\mathbf{p}_{f_j} - {}^{G}\mathbf{p}_{B_i}(t+t_d)\right) + {}^{C}\mathbf{p}_B \tag{4.56}$$

For a given feature track, the ${}^{G}\mathbf{p}_{f_j}$ is calculated by triangulation, where the camera poses are treated as a fixed constants and only the ${}^{G}\mathbf{p}_{f_j}$ is variable. Details of this procedure are described in section 3.3.3.

To form the residual vector for a single observation of the feature is formed by as a difference between observed position $\mathbf{z}_i^{(j)}$ as provided by the feature tracker and its predicted position $\hat{\mathbf{z}}_i^{(j)}$ obtained by reprojecting its triangulated position onto ith camera frame.

$$\mathbf{r}_i^{(j)} = \mathbf{z}_i^{(j)} - \hat{\mathbf{z}}_i^{(j)} \tag{4.57}$$

$$= \mathbf{z}_i^{(j)} - h\left({}^{C}_{B}\hat{\mathbf{R}}\,{}^{B_i}_{G}\hat{\mathbf{R}}(t+\hat{t}_d)\left({}^{G}\hat{\mathbf{p}}_{f_j} - {}^{G}\hat{\mathbf{p}}_{B_i}(t+\hat{t}_d)\right) + {}^{C}\hat{\mathbf{p}}_B\right) \tag{4.58}$$

With the residual defined, it needs to be linearized. It is done by applying the eq. (3.38) to the residual, leading to

$$\mathbf{r}_i^{(j)} \simeq \frac{\partial\, \hat{\mathbf{z}}_i^{(j)}}{\partial\, \tilde{\mathbf{X}}_k} \tilde{\mathbf{X}}_k + \frac{\partial\, \hat{\mathbf{z}}_i^{(j)}}{\partial\, {}^G\tilde{\mathbf{p}}_{f_j}} {}^G\tilde{\mathbf{p}}_{f_j} + \mathbf{n}_i^{(j)} \tag{4.59}$$

Because the residual does depend of the ${}^G\mathbf{p}_{f_j}$, it is used in the linearization as well.

By using the chain rule, the approximation can be modified as

$$\begin{aligned}\mathbf{r}_i^{(j)} \simeq & \left.\frac{\partial\, h\left({}^{C_i}\mathbf{p}_{f_j}\right)}{\partial\, {}^{C_i}\mathbf{p}_{f_j}}\right|_{{}^{C_i}\mathbf{p}_{f_j}={}^{C_i}\hat{\mathbf{p}}_{f_j}} \frac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, \tilde{\mathbf{X}}_k} \tilde{\mathbf{X}}_k \\ & + \left.\frac{\partial\, h\left({}^{C_i}\mathbf{p}_{f_j}\right)}{\partial\, {}^{C_i}\mathbf{p}_{f_j}}\right|_{{}^{C_i}\mathbf{p}_{f_j}={}^{C_i}\hat{\mathbf{p}}_{f_j}} \frac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, {}^G\tilde{\mathbf{p}}_{f_j}} {}^G\tilde{\mathbf{p}}_{f_j} \\ & + \mathbf{n}_i^{(j)}\end{aligned} \tag{4.60}$$

which can be rewritten as

$$\mathbf{r}_i^{(j)} \simeq \mathbf{H}_{\mathbf{X},i}^{(j)} \tilde{\mathbf{X}} + \mathbf{H}_{f,i}^{(j)} {}^G\tilde{\mathbf{p}}_{f_j} + \mathbf{n}_i^{(j)} \tag{4.61}$$

$$\mathbf{H}_{\mathbf{X},i}^{(j)} = \mathbf{J}_i^{(j)} \frac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, \tilde{\mathbf{X}}} \tag{4.62}$$

$$\mathbf{H}_{f,i}^{(j)} = \mathbf{J}_i^{(j)} \frac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, {}^G\tilde{\mathbf{p}}_{f_j}} \tag{4.63}$$

$$\mathbf{J}_i^{(j)} = \left.\frac{\partial\, h\left({}^{C_i}\mathbf{p}_{f_j}\right)}{\partial\, {}^{C_i}\mathbf{p}_{f_j}}\right|_{{}^{C_i}\mathbf{p}_{f_j}={}^{C_i}\hat{\mathbf{p}}_{f_j}} \tag{4.64}$$

Stacking the eqs. (4.61) for each observation leads to

$$\begin{bmatrix}\mathbf{r}_1^{(j)} \\ \mathbf{r}_2^{(j)} \\ \vdots \\ \mathbf{r}_{M_j}^{(j)}\end{bmatrix} = \begin{bmatrix}\mathbf{H}_{\mathbf{X},1}^{(j)} \\ \mathbf{H}_{\mathbf{X},2}^{(j)} \\ \vdots \\ \mathbf{H}_{\mathbf{X},M_j}^{(j)}\end{bmatrix} \tilde{\mathbf{X}} + \begin{bmatrix}\mathbf{H}_{f,1}^{(j)} \\ \mathbf{H}_{f,2}^{(j)} \\ \vdots \\ \mathbf{H}_{f,M_j}^{(j)}\end{bmatrix} {}^G\tilde{\mathbf{p}}_{f_j} + \begin{bmatrix}\mathbf{n}_1^{(j)} \\ \mathbf{n}_2^{(j)} \\ \vdots \\ \mathbf{n}_{M_j}^{(j)}\end{bmatrix} \tag{4.65}$$

$$\mathbf{r}^{(j)} = \mathbf{H}_{\mathbf{X}}^{(j)} \tilde{\mathbf{X}} + \mathbf{H}_f^{(j)} {}^G\tilde{\mathbf{p}}_{f_j} + \mathbf{n}^{(j)} \tag{4.66}$$

Since all the observations of a feature are mutually independent, vector $\mathbf{n}^{(j)}$ has the covariance matrix $\sigma_{\text{im}}^2 \mathbf{I}_{2M_j}$.

The model defined by (4.66) is not in the form required by the EKF (eq. (4.52)). That is because the vector ${}^G\tilde{\mathbf{p}}_{f_j}$ is correlated with the error-state $\tilde{\mathbf{X}}$. To address this issue, the residual vector $\mathbf{r}^{(j)}$ can be projected onto the left null space of the matrix $\mathbf{H}_f^{(j)}$. By letting the matrix $\mathbf{A}^{(j)}$ be a unitary matrix with columns corresponding to the left null space of the matrix $\mathbf{H}_f^{(j)}$, the residual vector can be redefined as

$$\mathbf{r}^{(j)} \simeq \mathbf{H}_{\mathbf{X}}^{(j)} \tilde{\mathbf{X}} + \mathbf{H}_f^{(j)} {}^G\tilde{\mathbf{p}}_{f_j} + \mathbf{n}^{(j)} \tag{4.67}$$

$$\mathbf{A}^{(j)\,T} \mathbf{r}^{(j)} \simeq \mathbf{A}^{(j)\,T} \mathbf{H}_{\mathbf{X}}^{(j)} \tilde{\mathbf{X}} + \mathbf{A}^{(j)\,T} \mathbf{n}^{(j)} \tag{4.68}$$

$$\mathbf{r}_o^{(j)} \simeq \mathbf{H}_o^{(j)} \tilde{\mathbf{X}} + \mathbf{n}_o^{(j)} \tag{4.69}$$

The matrix $\mathbf{H}_f^{(j)}$ of size $2M_j \times 3$ has a full column rank so its left nullspace has a rank of $2M_j - 3$. Because of this, the refined residual $\mathbf{r}_o^{(j)}$ is a vector of dimension $(2M_j - 3) \times 1$ and it is independent of the feature position error. The covariance matrix of the vector $\mathbf{n}_o^{(j)}$ is

$$\mathbb{E}\left[\mathbf{n}_o^{(j)}\mathbf{n}_o^{(j)\,T}\right] = \sigma_{\text{im}}^2 \mathbf{A}^{(j)\,T}\mathbf{A}^{(j)} = \sigma_{\text{im}}^2 \mathbf{I}_{2M_j-3} \tag{4.70}$$

4.6.2 Outlier rejection

Before updating the state using the feature residuals, a statistical test to separate the outliers can be employed. Two of the most common causes of outliers are tracking failures and observations of non-static features (i.e. cars and pedestrians).

Not all tracking failures result in outliers. For example, losing a track of a feature that is still visible only causes a suboptimal performance, not an outlier. A typical outlier is caused by appending an observation to an incorrect feature track. It will then continue to track this new feature instead of the original one.

For the purpose of the outlier rejection, we carry out the Mahalanobis gating test. The χ^2 statistics γ is generally calculated as

$$\gamma = \sum_{i=1}^{M_j} \left(\frac{X_i - \mu_i}{\sigma_i}\right)^2 \tag{4.71}$$

The residual $\mathbf{r}_o^{(j)}$ is already the difference between the observation and hypothesis. The covariance of the residual is given by $\mathbf{H}_o^{(j)}\mathbf{\Sigma}_{k+1|k}\mathbf{H}_o^{(j)\,T} + \sigma_{\text{im}}^2\mathbf{I}$

$$\gamma^{(j)} = \mathbf{r}_o^{(j)}\left(\mathbf{H}_o^{(j)}\mathbf{\Sigma}_{k+1|k}\mathbf{H}_o^{(j)\,T} + \sigma_{\text{im}}^2\mathbf{I}\right)^{-1}\mathbf{r}_o^{(j)\,T} \tag{4.72}$$

The value $\gamma^{(j)}$ is compared to the 95th percentile of the χ^2 distribution with $2M_j - 3$ degrees of freedom. If $\gamma^{(j)}$ is smaller than this threshold, the feature track of jth feature is considered an inlier.

4.6.3 Multi-feature measurement model

The eq. (4.69) provides the model for a single feature track. Assuming that L features are available for the residualization in a given step, elements of the eq. (4.69) can be stacked forming

$$\begin{bmatrix}\mathbf{r}_o^{(1)}\\ \mathbf{r}_o^{(2)}\\ \vdots\\ \mathbf{r}_o^{(L)}\end{bmatrix} = \begin{bmatrix}\mathbf{H}_o^{(1)}\\ \mathbf{H}_o^{(2)}\\ \vdots\\ \mathbf{H}_o^{(L)}\end{bmatrix}\tilde{\mathbf{X}} + \begin{bmatrix}\mathbf{n}_o^{(1)}\\ \mathbf{n}_o^{(2)}\\ \vdots\\ \mathbf{n}_o^{(L)}\end{bmatrix} \tag{4.73}$$

$$\mathbf{r}_o = \mathbf{H}_{\mathbf{X}}\tilde{\mathbf{X}} + \mathbf{n}_o \tag{4.74}$$

The size of the vector $\mathbf{r}_o$ is $d \times 1$, $d = \sum_{j=1}^{L}(2M_j - 3)$. Since all the feature measurements are independent, $\mathbf{n}_0$ has a covariance matrix $\mathbf{R}_o = \sigma_{\text{im}}^2\mathbf{I}_d$.

4.6.4 Residual marginalization

Consider an example of 10 features being observed, each by 10 cameras. Then $d = 170$, which is quite large and can cause performance issues. For those cases where d is bigger than the size of the error-state, a QR decomposition can be employed which will keep the matrices smaller.

$$\mathbf{H}_{\mathbf{X}} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tag{4.75}$$

with $\mathbf{Q}_1$ and $\mathbf{Q}_2$ being unitary matrices and $\mathbf{T}_H$ being an upper triangular matrix. Rewriting the eq. (4.74) leads to

$$\mathbf{r}_o = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{X}} + \mathbf{n}_0 \tag{4.76}$$

$$\begin{bmatrix} \mathbf{Q}_1^T \mathbf{r}_o \\ \mathbf{Q}_2^T \mathbf{r}_o \end{bmatrix} = \begin{bmatrix} \mathbf{T}_H \\ \mathbf{0} \end{bmatrix} \tilde{\mathbf{X}} + \begin{bmatrix} \mathbf{Q}_1^T \mathbf{n}_0 \\ \mathbf{Q}_2^T \mathbf{n}_0 \end{bmatrix} \tag{4.77}$$

The residual $\mathbf{Q}_2^T \mathbf{r}_o$ consists of the noise only and is independent of the error-state, thus it can be omitted and the EKF-update can be done with the residual of

$$\mathbf{r}_n = \mathbf{Q}_1^T \mathbf{r}_o = \mathbf{T}_H \tilde{\mathbf{X}} + \mathbf{n}_n \tag{4.78}$$

where $\mathbf{n}_n = \mathbf{Q}_1^T \mathbf{n}_0$ is the noise vector with a covariance matrix of $\mathbf{R}_n = \mathbf{Q}_1^T \mathbf{R}_o \mathbf{Q}_1 = \sigma_{\text{im}}^2 \mathbf{I}_r$ where r is the number of columns of matrix $\mathbf{Q}_1$.

4.6.5 EKF update

The EKF update can be done using either non-marginalized (eq. (4.74)) or marginalized (eq. (4.78)) measurement model. In this work, the latter one is used.

The Kalman gain is given by

$$\mathbf{K} = \boldsymbol{\Sigma}_{k+1|k} \mathbf{T}_H^T \left(\mathbf{T}_H \boldsymbol{\Sigma}_{k+1|k} \mathbf{T}_H^T + \mathbf{R}_n \right)^{-1} \tag{4.79}$$

and the correction of the state together with the new covariance matrix is

$$\hat{\tilde{\mathbf{X}}} = \mathbf{K} \mathbf{r}_n \tag{4.80}$$

$$\boldsymbol{\Sigma}_{k+1|k+1} = \left(\mathbf{I} - \mathbf{K} \mathbf{T}_H \right) \boldsymbol{\Sigma}_{k+1|k} \left(\mathbf{I} - \mathbf{K} \mathbf{T}_H \right)^T + \mathbf{K} \mathbf{R}_n \mathbf{K}^T \tag{4.81}$$

4.6.6 Injection

When the estimate of the error-state $\hat{\tilde{\mathbf{X}}}$ is obtained, it is injected into the nominal state

$$\hat{\mathbf{X}}_{k+1|k+1} \leftarrow \hat{\mathbf{X}}_{k+1|k} \oplus \hat{\tilde{\mathbf{X}}} \tag{4.82}$$

The operator $\oplus$ denotes a general composition operator. For all elements but quaternions, a standard addition is used e.g. ${}^G\hat{\mathbf{p}}_B \leftarrow {}^G\hat{\mathbf{p}}_B + {}^G\hat{\tilde{\mathbf{p}}}_B$.

In the error-state, quaternions are expressed in the form of small rotation vectors. Before being injected into the nominal state, they first need to be approximated using the small quaternion approximation (eq. (A.17)). The injection of ${}^{B}_{G}\hat{\mathbf{q}}$ is

$$ {}^{B}_{G}\hat{\mathbf{q}} \leftarrow \begin{bmatrix} \frac{1}{2}{}^{B}_{G}\hat{\tilde{\mathbf{q}}} \\ 1 \end{bmatrix} \otimes {}^{B}_{G}\hat{\mathbf{q}} \tag{4.83} $$

with $\otimes$ being a standard quaternion multiplication.

In order to set the mean of the error-state to the zero vector, the operation

$$ \hat{\tilde{\mathbf{X}}} \leftarrow \tilde{\mathbf{X}} \ominus \Delta\hat{\tilde{\mathbf{X}}} \tag{4.84} $$

should be carried out. The operator $\ominus$ is the inverse operation to $\oplus$ used for the injection. That leads to the reset operation of

$$ \hat{\tilde{\mathbf{X}}} \leftarrow \mathbf{0} \tag{4.85} $$

$$ \mathbf{\Sigma}_{k+1|k+1} \leftarrow \mathbf{G}\mathbf{\Sigma}_{k+1|k+1}\mathbf{G}^{T} \tag{4.86} $$

$$ \tag{4.87} $$

where

$$ \mathbf{G} = \frac{\partial\left(\tilde{\mathbf{X}} \ominus \Delta\hat{\tilde{\mathbf{X}}}\right)}{\partial\tilde{\mathbf{X}}} \tag{4.88} $$

The value of $\mathbf{G}$ is close to being an identity matrix and thus this operation is omitted for performance reasons.

4.6.7 Pruning

If the number of body poses in the state N is equal to the upper bound $N_{\max}$, at least one body pose needs to be removed from the state. When doing so, the covariance matrix needs to be adjusted accordingly by removing corresponding rows and columns.

Mourikis and Roumeliotis [19] chose to prune $N_{\max}/3$ poses, evenly distributed in time, starting from the second oldest one. They reasoned that older poses typically posses a wider baseline between camera frames and thus provide a more valuable information. They claim that this approach works well in practice. Their experimental data was collected by a camera and IMU mounted on a car.

On the other hand, Shelley [62] opted to only cut the oldest camera pose. As shown on the evaluated dataset, the accuracy of the system was not degraded. The dataset was collected by a hand-held device.

An arbitrary pruning schema can be used. The best performing schema might be even dependent on the type of dataset, for example amount of shaking, vibrations or speed.

4.7 Algorithm summary

To help better understand how the algorithm works, it is summarized in Algorithm 4, including all the relevant functions.

Algorithm 4 A Multi-State Constrained Kalman Filter

b = EMPTYBUFFER()
loop
  m = GETMEASUREMENT()
  if m is an IMU measurement **then**
    $b \leftarrow$ APPENDTOBUFFER(b, m)
  else
    t = MEASUREMENTTIME(m)
    feature tracks = TRACKFEATURES(m)
    PROPAGATE(b, $t + t_d$)
    AUGMENT()
    UPDATE(feature tracks)
    PRUNE()
  end if
end loop

Algorithm 5 MSCKF: Propagation to time t

procedure PROPAGATE(b, t)
  while filter time $\leq t$ **do**
    Pop the first measurement $(t_m, \boldsymbol{\omega}_m, \mathbf{a}_m)$ from the buffer b.
    Calculate $\mathbf{F}$ according to eq. (4.22) and $\mathbf{G}$ according to eq. (4.22)
    Propagate ${}^B_G\mathbf{q}$ as described in section 4.4.1.
    Propagate ${}^G\mathbf{v}_B$ and ${}^G\mathbf{p}_B$ as described in section 4.4.2.
    Calculate $\boldsymbol{\Phi}_k$ and $\mathbf{Q}_d$ according to 4.46.
    $\boldsymbol{\Sigma} \leftarrow \boldsymbol{\Phi}_k \boldsymbol{\Sigma} \boldsymbol{\Phi}_k^T + \mathbf{Q}_d$ ▷ Covariance matrix propagation
    Set the filter time to t_m.
  end while
end procedure

Algorithm 6 MSCKF: State augmentation

procedure AUGMENT()
  $\boldsymbol{\pi}_{B_i} = \begin{bmatrix} {}^B_G\bar{\mathbf{q}}^T & {}^G\mathbf{p}_B^T & {}^G\mathbf{v}_B^T \end{bmatrix}^T$
  $\hat{\mathbf{X}}_k \leftarrow \begin{bmatrix} \hat{\mathbf{X}}_k^T & \boldsymbol{\pi}_{B_i}^T \end{bmatrix}^T$
  $\mathbf{J}_{\boldsymbol{\pi}} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \dots \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \dots \end{bmatrix}$
  $\boldsymbol{\Sigma}_k \leftarrow \begin{bmatrix} \mathbf{I}_{22+6*N} \\ \mathbf{J}_{\boldsymbol{\pi}} \end{bmatrix} \boldsymbol{\Sigma}_k \begin{bmatrix} \mathbf{I}_{22+6*N} \\ \mathbf{J}_{\boldsymbol{\pi}} \end{bmatrix}^T$
end procedure

Algorithm 7 MSCKF: Update step

procedure UPDATE(feature tracks)
tracks to residualize ← GETFEATURETRACKSTORESIDUALIZE(feature tracks)
$\mathbf{r}_o = \emptyset, \quad \mathbf{H}_\mathbf{X} = \emptyset$
for jth feature track in tracks to residualize **do**
$\mathbf{r}_o^{(j)}, \mathbf{H}_o^{(j)}$ = RESIDUALIZEFEATURETRACK(jth feature track)
$\mathbf{r}_o$ = STACK($\mathbf{r}_o$, $\mathbf{r}_o^{(j)}$)
$\mathbf{H}_\mathbf{X}$ = STACK($\mathbf{H}_\mathbf{X}$, $\mathbf{H}_o^{(j)}$)
end for
if $\mathbf{r}_o$ is larger than error-state **then**
$\mathbf{Q}, \mathbf{R}$ = QRDECOMPOSITION($\mathbf{H}_\mathbf{X}$)
Get $\mathbf{T}_H$ and $\mathbf{Q}_1$ from $\mathbf{Q}$ and $\mathbf{R}$ according to eq. (4.75)
$\mathbf{r}_n = \mathbf{Q}_1^T \mathbf{r}_o$
EKFUPDATE($\mathbf{r}_n$, $\mathbf{T}_H$)
else
EKFUPDATE($\mathbf{r}_o$, $\mathbf{H}_\mathbf{X}$)
end if
end procedure

Algorithm 8 MSCKF: Residualize feature track

procedure RESIDUALIZEFEATURETRACK(jth feature track)
${}^G\hat{\mathbf{p}}_{f_j}$ ← TRIANGULATE($\hat{\mathbf{X}}_k$, jth feature track)
$\mathbf{r}^{(j)} = \emptyset, \quad \mathbf{H}_\mathbf{X}^{(j)} = \emptyset, \quad \mathbf{H}_f^{(j)} = \emptyset$
for ith observation of jth feature track **do**
$\mathbf{r}_i^{(j)}, \mathbf{H}_{\mathbf{X},i}^{(j)}, \mathbf{H}_{f,i}^{(j)}$ ← RESIDUALIZEOBSERVATION(${}^G\hat{\mathbf{p}}_{f_j}$, $\mathbf{z}_i^{(j)}$)
$\mathbf{r}^{(j)}$ ← STACK($\mathbf{r}^{(j)}$, $\mathbf{r}_i^{(j)}$)
$\mathbf{H}_\mathbf{X}^{(j)}$ ← STACK($\mathbf{H}_\mathbf{X}^{(j)}$, $\mathbf{H}_{\mathbf{X},i}^{(j)}$)
$\mathbf{H}_f^{(j)}$ ← STACK($\mathbf{H}_f^{(j)}$, $\mathbf{H}_{f,i}^{(j)}$)
end for
Calculate $\mathbf{A}^{(j)}$ as in section 4.6.1
$\mathbf{r}_o^{(j)} \leftarrow \mathbf{A}^{(j),T} \mathbf{r}^{(j)}$
$\mathbf{H}_o^{(j)} \leftarrow \mathbf{A}^{(j),T} \mathbf{H}_\mathbf{X}^{(j)}$
return $\mathbf{r}_o^{(j)}$, $\mathbf{H}_o^{(j)}$
end procedure

Algorithm 9 MSCKF: Residualize observation

procedure ResidualizeObservation($^{G}\hat{\mathbf{p}}_{f_j}$, $\mathbf{z}_i^{(j)}$)

$^{C_i}\hat{\mathbf{p}}_{f_j} = {}^{C}_{B}\hat{\mathbf{R}}\,{}^{B_i}_{G}\hat{\mathbf{R}}(t+\hat{t}_d)\left({}^{G}\hat{\mathbf{p}}_{f_j} - {}^{G}\hat{\mathbf{p}}_{B_i}(t+\hat{t}_d)\right) + {}^{C}\hat{\mathbf{p}}_{B}$

$\mathbf{r}_i^{(j)} = \mathbf{z}_i^{(j)} - h\left({}^{C_i}\hat{\mathbf{p}}_{f_j}\right)$

$\mathbf{H}_{\mathbf{X},i}^{(j)} = \mathbf{J}_i^{(j)} \dfrac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, \tilde{\mathbf{X}}}$

$\mathbf{H}_{f,i}^{(j)} = \mathbf{J}_i^{(j)} \dfrac{\partial\, {}^{C_i}\mathbf{p}_{f_j}}{\partial\, {}^{G}\tilde{\mathbf{p}}_{f_j}}$

$\mathbf{J}_i^{(j)} = \left.\dfrac{\partial\, h\left({}^{C_i}\mathbf{p}_{f_j}\right)}{\partial\, {}^{C_i}\mathbf{p}_{f_j}}\right|_{{}^{C_i}\mathbf{p}_{f_j} = {}^{C_i}\hat{\mathbf{p}}_{f_j}}$

return $\mathbf{r}_i^{(j)}$, $\mathbf{H}_{\mathbf{X},i}^{(j)}$, $\mathbf{H}_{f,i}^{(j)}$

end procedure

Algorithm 10 MSCKF: EKF Update

procedure EkfUpdate($\mathbf{r}_o$, $\mathbf{H}_{\mathbf{X}}$)

$\mathbf{K} = \boldsymbol{\Sigma}_{k+1|k}\mathbf{T}_H^T\left(\mathbf{T}_H\boldsymbol{\Sigma}_{k+1|k}\mathbf{T}_H^T + \mathbf{R}_n\right)^{-1}$

$\hat{\tilde{\mathbf{X}}} = \mathbf{K}\mathbf{r}_n$

$\hat{\mathbf{X}}_{k+1|k+1} = \hat{\mathbf{X}}_{k+1|k} \oplus \hat{\tilde{\mathbf{X}}}$ ▷ The operator $\oplus$ is described in section 4.6.6.

$\boldsymbol{\Sigma}_{k+1|k+1} = (\mathbf{I} - \mathbf{K}\mathbf{T}_H)\,\boldsymbol{\Sigma}_{k+1|k}\,(\mathbf{I} - \mathbf{K}\mathbf{T}_H)^T + \mathbf{K}\mathbf{R}_n\mathbf{K}^T$

end procedure

5. Fusion with a GPS

As discussed at the beginning of the work (section 4.1), the position and yaw are not observable by any VIO algorithm, including MSCKF. This will lead to an unlimited drift of both quantities. To provide a reliable, lifelong localization there is a need for a system that provides a measurement which makes them observable.

This chapter presents a way how the GPS can be used to reduce the position drift of MSCKF. This is one of the contributions of this work.

The principles of GPS measurements are described in the first section. The second section describes different geodetic coordinate frames, which are then used to derive the GPS measurement model. In the end of this chapter, a fusion of such model with MSCKF is shown.

5.1 GPS sensor

GPS is a sensor that measures the distance from a satellite at known position in Earth's orbit, to determine the sensor position.

The term GPS is commonly used to refer to the more general category of global navigation satellite system (GNSS). The GPS was the first such system. During the decades of its existence, other constellations emerged. Russian GLONASS, European Galileo and Chinese BeiDou-2 are the ones with global coverage. A few other regional systems exist. Most of the GPS sensors can in fact work with multiple or all other GNSSs previously mentioned.

At a very simplified level, a GPS sensor (receiver) listens to the signal transmitted from the satellite. The signal contains the reading from the on-board atomic clock. The receiver then measures the time it took for the signal to arrive to the antenna. From this *time-of-flight* measurement, a distance is calculated. This distance is called a *pseudorange*. A single pseudorange is not enough to determine the exact location of the receiver. At least 4 pseudoranges are needed to do that. When more pseudoranges are available, the measurement uncertainty might be lower. A more in-depth description can be found in many books dedicated to this topic, example of which is a book by Parkinson et al. [63].

The result of the localization is the position estimate of the antenna. The measurement is composed of several pieces of information. It contains a latitude, longitude (both in degrees) and altitude (in meters) in WGS84 coordinate frame, which is described in section 5.2.1. It also includes current GPS time[1] and measurement uncertainty. The uncertainty is commonly expressed as two numbers, the horizontal and vertical uncertainty, both measured in meters.

GPS has some inherent limitations, causing imperfections of the measurements. For example when the receiver is in the valley, caused by urban buildings or natural obstacles, only the satellites from one area of the sky are visible, causing the underlaying linear system to be poorly conditioned. Other significant examples are ionospheric distortions. The ionosphere is the outermost layer of

[1]The GPS time was set to the value of the UTC time at the midnight of the 6th of January, 1980. Since then no leap seconds were added to the GPS time and thus is 27 seconds behind the UTC time now.

the Earth's atmosphere. It contains charged ions, which curve the trajectory of the signal from the satellite, skewing the measurement.

Accuracy obtained by a commodity receiver can be as good as 5 meters. Many, commonly very expensive, receivers can achieve accuracy of up to a few centimeters. They utilize and often combine together various sophisticated systems such as L5 user segment [64], differential GPS, SBAS or fusion with an accelerometer. Because of this, receivers usually vary in their noise characteristics, which might be utilized when designing a system for one particular receiver.

5.2 Geodetic coordinate frames

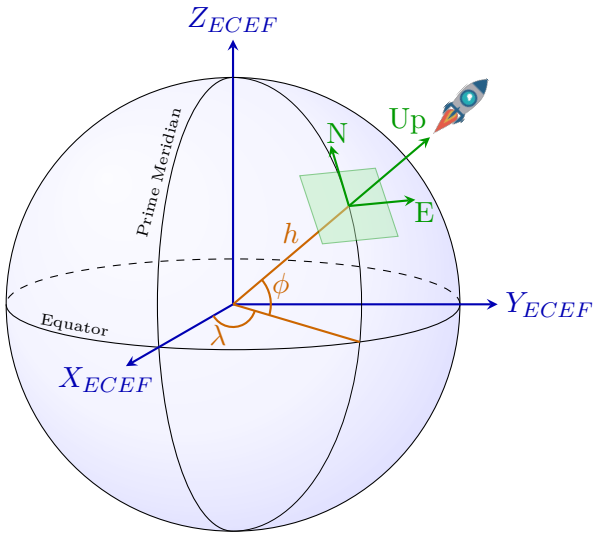


Figure 5.1: Depiction of multiple coordinate frames and their relation to the Earth and to each other. The ENU frame is the tangential plane at point ϕ, λ, h.

5.2.1 WGS84

The World Geodetic System defines the geodetic reference system. The WGS84 is its revision from 1984. It defines a coordinate frame of the Earth whose origin is located at the gravitational center of mass of the planet (when including the gravity of oceans and atmosphere). The axes are aligned such that the Z-axis points toward the North pole, X-axis is the intersection of the Prime meridian and the plane defined by the equator and the Y-axis is defined such that the coordinate frame is orthogonal and right-handed[2].

Any point can be then described by the angles from the Equator (ϕ) and the Prime Meridian (λ) and its altitude (h) (relative to the sea-level).

Most notably, GPS satellites are using this standard and also the measurements are represented in terms of it.

[2]The provided description is only approximate and the true definition of Z and X axes differ. This simplification is good enough for the purpose of this work.

5.2.2 ECEF

Any point described by its latitude, longitude and altitude in WGS84 can be converted to corresponding XYZ values. The ECEF reference frame is aligned with the WGS84 frame but its units are expressed in meters from the origin instead of the angles and altitude.

5.2.3 ENU

When working inside of a limited area, it is easier to neglect the Earth's curvature and approximate the world by a tangential plane. One of the commonly used reference frames is ENU. It is a left-handed coordinate system such that at its origin, X-axis points towards east, Y-axis towards north and Z-axis points up. There are infinitely many of such frames and they are dependent on where the tangent plane is chosen to be. Fig. (5.1) shows an example ENU frame on the surface of the Earth.

The counterpart reference frame to ENU is NED, where X, Y and Z axes point towards the north, east and down. This system is right-handed and should not be mixed with ENU as they are different but used in similar situations.

5.3 Conversion

A conversion between some of the frames are possible. They can be seen in the diagram (5.2).

Figure 5.2: Available conversions between geodetic coordinate frames.

5.3.1 WGS84 to ECEF

To convert the geodetic coordinates (WGS84) ϕ, λ, h to the ECEF frame, the following formula can be used

$$
{}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}\left(\phi,\lambda,h\right) = \begin{bmatrix} (N(\phi)+h)\cos\phi\cos\lambda \\ (N(\phi)+h)\cos\phi\sin\lambda \\ \left(\frac{a^2}{b^2}N(\phi)+h\right)\sin\phi \end{bmatrix} \tag{5.1}
$$

where

$$
N(\phi) = \frac{a^2}{\sqrt{a^2\cos^2\phi + b^2\sin^2\phi}} \tag{5.2}
$$

$$
a = 6378137 \tag{5.3}
$$

$$
b = a * (1 - f) \tag{5.4}
$$

$$
f = 1/298.2572235630 \tag{5.5}
$$

The parameters a, b, f represent the semi-major axis, semi-minor axis and the flattening of the reference ellipsoid defined by WGS84.

5.3.2 ECEF to WGS84

The conversion from ECEF to WGS84, realized by the function ${}^{\text{WGS84}}_{\text{ECEF}}\mathbf{f}(\cdot)$, is non-trivial and involves an iterative minimization algorithm. Details of the conversion can be found in literature [65, 66].

5.3.3 ENU to ECEF

In order to compute the conversion from ENU to ECEF frame, the origin of the ENU frame given by coordinates ϕ_0, λ_0, h_0 must be known.

$$ {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left({}^{\text{ENU}}\mathbf{p}_x, \phi_0, \lambda_0, h_0\right) = \mathbf{A}\left(\phi, \lambda\right) {}^{\text{ENU}}\mathbf{p}_x + {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}\left(\phi_0, \lambda_0, h_0\right) \tag{5.6} $$

where

$$ \mathbf{A}\left(\phi, \lambda\right) = \begin{bmatrix} -\sin\lambda & -\sin\phi\cos\lambda & \cos\phi\cos\lambda \\ \cos\lambda & -\sin\phi\sin\lambda & \cos\phi\sin\lambda \\ 0 & \cos\phi & \sin\phi \end{bmatrix} \tag{5.7} $$

5.3.4 ECEF to ENU

The function ${}^{\text{ENU}}_{\text{ECEF}}\mathbf{f}(\cdot)$, converting a point from EFEC to ENU coordinate system can be obtained by inverting the function ${}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}(\cdot)$ and is omitted.

5.4 Measurement model

To fuse GPS with the filter, there is a need for a constraint in the same form as for the camera. That means defining a measurement model function $\mathbf{h}_{\text{GPS}}$ such that

$$ \mathbf{r}_k^{(\text{GPS})} = \mathbf{z}_k^{(\text{GPS})} - \hat{\mathbf{z}}_k^{(\text{GPS})} \tag{5.8} $$

$$ = \mathbf{z}_k^{(\text{GPS})} - \mathbf{h}_{\text{GPS}}\left(\mathbf{X}_k\right) \tag{5.9} $$

The MSCKF keeps track of the position of the body relative to the global frame. The global frame, despite the fact that the name might suggest otherwise, is unrelated to any absolute location in the world. To be able to formulate the function $\mathbf{h}_{\text{GPS}}$, this relationship must be captured by adding new variables to the state. The exact variables needed depend on the particular function $\mathbf{h}_{\text{GPS}}$.

There are multiple candidates for the function $\mathbf{h}_{\text{GPS}}$, each of which has a different meaning of the residuum vector $\mathbf{r}_k^{(\text{GPS})}$. Next, a few of those candidates are presented with a brief discussion of the possible issues linked to each of them.

5.4.1 Residuum in WGS84 coordinate frame

Perhaps the most intuitive way of defining the residuum would be to express it in WGS84 coordinate frame, as the raw measurement from the GPS $\mathbf{z}_k^{(GPS)}$ is expressed in it too.

In this case, the filter would need to include the full 6 degrees of freedom transformation from $\{G\}$ to the ECEF frame, which can then be converted to the WGS84. The residuum is then

$$\mathbf{r}_k^{(\text{GPS})} = \mathbf{z}_k^{(\text{GPS})} - \hat{\mathbf{z}}_k^{(\text{GPS})} \tag{5.10}$$

$$= \mathbf{z}_k^{(\text{GPS})} - {}^{\text{WGS84}}_{\text{ECEF}}\mathbf{f}\left({}^{\text{ECEF}}_{G}\mathbf{R}\left({}^{G}\mathbf{p}_B - {}^{G}\mathbf{p}_{\text{ECEF}}\right)\right) \tag{5.11}$$

where ${}^{\text{ECEF}}_{G}\mathbf{R}$ and ${}^{G}\mathbf{p}_{\text{ECEF}}$ describe the rotation and position of the frame $\{G\}$ in the ECEF frame. Both of these would need to be added to the state (the rotation matrix would be stored as a quaternion ${}^{\text{ECEF}}_{G}\mathbf{q}$).

Despite being a very intuitive solution, there are at least three drawbacks of such approach.

First, when differentiating the Jacobian of the $\mathbf{h}_{\text{GPS}}$ function, it would be necessary to find the Jacobian of function ${}^{\text{WGS84}}_{\text{ECEF}}\mathbf{f}(\cdot)$ which is an iterative algorithm. This can be solved by either using a fixed number of iterations and calculating the Jacobian of all the steps or by utilizing the Jacobian of function ${}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\cdot)$ and the theorem about derivative of the inverse function.

Second, the transformation from the global frame to ECEF is expressed with 6 degrees of freedom. As will be shown later in section 5.4.3, the underlaying problem has in fact only 4 degrees of freedom. Since this approach is overparametrized it would result in ill-behaved convergence of the parameters.

Third, and perhaps the most important issue is related to the numerical stability of the problem. For example, if a robot is approximately at coordinates $\phi = 50.088°$, $\lambda = 14.403°$, $h = 200$ and the residual resembles an error of 1 cm, the norm of the residual is roughly $-2.43 * 10^{-9}$ whereas ${}^{G}\mathbf{p}_{\text{ECEF}} \simeq 10^6 * \begin{bmatrix}-2.34 & 3.37 & 4.87\end{bmatrix}^T$. These values are roughly 15 orders of magnitude apart which causes significant numerical instabilities even when working with 64bit IEEE float numbers.

5.4.2 Residuum in ENU coordinate frame

Another possible solution is to fix an arbitrary ENU frame in advance (i.e. by fixing the parameters ϕ_0, λ_0, h_0 beforehand or by using the first GPS measurement) and estimate the transformation from global to the ENU frame. The residuum would then be

$$\mathbf{r}_k^{(\text{GPS})} = \mathbf{z}_k^{(\text{GPS})} - \hat{\mathbf{z}}_k^{(\text{GPS})} \tag{5.12}$$

$$= {}^{\text{ENU}}_{\text{ECEF}}\mathbf{f}\left({}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}\left(\mathbf{z}_k^{(\text{GPS})}\right), \phi_0, \lambda_0, h_0\right) - {}^{\text{ENU}}_{G}\mathbf{R}\left({}^{G}\mathbf{p}_B - {}^{G}\mathbf{p}_{\text{ENU}}\right) \tag{5.13}$$

Because the parameters ϕ_0, λ_0, h_0 are independent of the filter state, it is valid to apply such a transformation to the measurement. This approach does not suffer from numerical instabilities as the previous one but it still suffers from being overparametrized. In addition, one would need to define the extra parameters ϕ_0, λ_0, h_0 which can be avoided, simplifying the initial configuration process.

5.4.3 Residuum in ECEF coordinate frame

When defining the residuum in ECEF frame, all the issues mentioned before are either not present or can be easily avoided.

At first, a particular ENU frame needs to be chosen. One choice is particularly beneficial for the fusion process. That is the ENU frame with the origin incidental to the origin of the global frame $\{G\}$. The advantages are twofold. First, the translation between the ENU frame and the global frame becomes zero, simplifying the equations involved.

To understand the second reason, an extra property of the global frame needs to be understood. Because of eq. (4.34), the gravity points in the reverse direction of the z-axis of the global frame. The same holds for the ENU frame since by definition, the z-axis points up (opposite direction of the gravity). So in order to align the two, only the yaw (orientation around the gravity) needs to be estimated.

In total, there are four parameters to be estimated by the filter, in order to predict the GPS measurement in the global frame. Namely, latitude (ϕ), longitude (λ) and altitude (h) of the origin of the global frame and its yaw (ψ). The residual vector is then

$$\mathbf{r}_k^{(\mathrm{GPS})} = \mathbf{z}_k^{(\mathrm{GPS})} - \hat{\mathbf{z}}_k^{(\mathrm{GPS})} \tag{5.14}$$

$$= {}^{\mathrm{ECEF}}_{\mathrm{WGS84}}\mathbf{f}\left(\mathbf{z}_k^{(\mathrm{GPS})}\right) - {}^{\mathrm{ECEF}}_{\mathrm{ENU}}\mathbf{f}\left(\mathbf{R}_z\left(\psi\right){}^{G}\mathbf{p}_B, \phi, \lambda, h\right) \tag{5.15}$$

where $\mathbf{R}_z\left(\psi\right)$ is the rotation matrix around the z-axis

$$\mathbf{R}_z\left(\psi\right) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.16}$$

5.5 Fusion with MSKCF

When a particular measurement model is chosen, it can be used to fuse the GPS information with the filter. This section shows the steps of the process by using the residuum in ECEF frame (section 5.4.3).

5.5.1 Timing

Because GPS measurements usually arrive at lower frequency than camera images, a standard update step is carried out if there is no new GPS measurement available. When the GPS sensor delivers a new measurement, it is processed together with the camera update. This neglects the time synchronization issues of the two sensors but GPS noise is usually high and will dominate all the noises involved in the update procedure, including timing issues.

When the GPS used is particularly accurate, or the robot is moving fast (the distance traveled between two camera frames is significant, compared to the GPS measurement noise), this issue should be addressed by performing a GPS-only update at the time of each measurement. It is also possible to estimate the sensor delay relative to the IMU, similarly as t_d for the camera.

5.5.2 Antenna alignment

GPS measures the position of the antenna relative to the world. In case of smartphones, the distance between the body frame (IMU) position and the antenna can be neglected as it is small comparing to the measurement noise. For cases where the GPS sensor is so accurate that this does not hold, or the antenna is mounted further away from the IMU, their relative position must be calibrated in advance and measurement model need to be adjusted accordingly. Alternatively, it can be online estimated by the algorithm, similar to how the camera-to-IMU spatial transformation is.

5.5.3 GPS measurement model

Based on the previous section, the residual vector for a GPS measurement can most beneficially be defined by eq. (5.15), leading to the measurement model

$$\mathbf{h}_{\text{GPS}}\left(\mathbf{X}\right) = {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{R}_z\left(\psi\right){}^{G}\mathbf{p}_B, \phi, \lambda, h\right) \tag{5.17}$$

Before it can be fused with the MSCKF, a Jacobian of it in form of eq. (4.52) needs to be found

$$\mathbf{r}_k^{(\text{GPS})} = \mathbf{H}_k^{(\text{GPS})}\tilde{\mathbf{X}}_k + \text{noise} \tag{5.18}$$

where the noise vector must be Gaussian and independent of the filter state.

The matrix $\mathbf{H}_k^{(\text{GPS})}$ is a $3 \times 26 + 6N$ matrix of where the only non-zero block correspond to the derivative of the function $\mathbf{h}_{\text{GPS}}$ by its parameters.

$$\frac{\partial\, \mathbf{h}_{\text{GPS}}\left(\mathbf{X}\right)}{\partial\, {}^{G}\tilde{\mathbf{p}}_B} = \left.\frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{p}, \hat{\phi}, \hat{\lambda}, \hat{h}\right)}{\partial\, \mathbf{p}}\right|_{\mathbf{p}=\mathbf{R}_z\left(\hat{\psi}\right){}^{G}\hat{\mathbf{p}}_B} \mathbf{R}_z\left(\hat{\psi}\right) \tag{5.19}$$

$$\frac{\partial\, \mathbf{h}_{\text{GPS}}\left(\mathbf{X}\right)}{\partial \begin{bmatrix}\tilde{\phi} & \tilde{\lambda} & \tilde{h}\end{bmatrix}} = \frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{R}_z\left(\hat{\psi}\right){}^{G}\hat{\mathbf{p}}_B, \hat{\phi}, \hat{\lambda}, \hat{h}\right)}{\partial \begin{bmatrix}\hat{\phi} & \hat{\lambda} & \hat{h}\end{bmatrix}} \tag{5.20}$$

$$\frac{\partial\, \mathbf{h}_{\text{GPS}}\left(\mathbf{X}\right)}{\partial\, \tilde{\psi}} = \frac{\partial\, \mathbf{h}_{\text{GPS}}\left(\mathbf{X}\right)}{\partial \begin{bmatrix}\tilde{\phi} & \tilde{\lambda} & \tilde{h}\end{bmatrix}} \mathbf{R}_z\left(\hat{\psi}\right){}^{G}\hat{\mathbf{p}}_B \tag{5.21}$$

For this, the derivative of the function ${}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}$ must be found.

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{p}, \phi, \lambda, h\right)}{\partial\, \mathbf{p}} = \mathbf{A}\left(\phi, \lambda\right) \tag{5.22}$$

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{p}, \phi, \lambda, h\right)}{\partial\, \phi} = \mathbf{M}_\phi\, \mathbf{p} + \frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi, \lambda, h)}{\partial\, \phi} \tag{5.23}$$

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{p}, \phi, \lambda, h\right)}{\partial\, \lambda} = \mathbf{M}_\lambda\, \mathbf{p} + \frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi, \lambda, h)}{\partial\, \lambda} \tag{5.24}$$

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{ENU}}\mathbf{f}\left(\mathbf{p}, \phi, \lambda, h\right)}{\partial\, h} = \frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi, \lambda, h)}{\partial\, h} \tag{5.25}$$

$$\mathbf{M}_\phi = \begin{bmatrix} 0 & -\cos\phi\cos\lambda & -\sin\phi\cos\lambda \\ 0 & -\cos\phi\sin\lambda & -\sin\phi\sin\lambda \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \quad \mathbf{M}_\lambda = \begin{bmatrix} -\cos\lambda & \sin\phi\sin\lambda & -\cos\phi\sin\lambda \\ -\sin\lambda & -\sin\phi\cos\lambda & \cos\phi\cos\lambda \\ 0 & 0 & 0 \end{bmatrix}$$

The Jacobian of the ${}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}$ is

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi,\lambda,h)}{\partial\,\phi} = \begin{bmatrix} \left(\frac{\partial\, N(\phi)}{\partial\,\phi}\cos\phi - (h+N(\phi))\sin\phi\right)\cos\lambda \\ \left(\frac{\partial\, N(\phi)}{\partial\,\phi}\cos\phi - (h+N(\phi))\sin\phi\right)\sin\lambda \\ \frac{1}{a^2}\left(b^2\sin\phi\frac{\partial\, N(\phi)}{\partial\,\phi} + (a^2h + b^2N(\phi))\cos\lambda\right) \end{bmatrix} \tag{5.26}$$

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi,\lambda,h)}{\partial\,\lambda} = \begin{bmatrix} -(h+N(\phi))\cos\phi\sin\lambda \\ (h+N(\phi))\cos\phi\cos\lambda \\ 0 \end{bmatrix} \tag{5.27}$$

$$\frac{\partial\, {}^{\text{ECEF}}_{\text{WGS84}}\mathbf{f}(\phi,\lambda,h)}{\partial\, h} = \begin{bmatrix} \cos\phi\cos\lambda \\ \cos\phi\sin\lambda \\ \sin\phi \end{bmatrix} \tag{5.28}$$

And finally the jacobian of the function $N(\phi)$ is

$$\frac{\partial\, N(\phi)}{\partial\,\phi} = \frac{\sqrt{2}\left(\frac{1}{f^2} - \left(\frac{1}{f}-1\right)^2\right)\frac{a}{f}\sin(2\phi)}{\sqrt{\left(\frac{2}{f^2} + \frac{2\cos(2\phi)}{f} - \frac{2}{f} - \cos(2\phi) + 1\right)^3}} \tag{5.29}$$

To form the measurement model in terms of eq. (5.18) the noise parameters of GPS must be understood. The sensor itself provides an estimate of the noise, which is calculated based on a number of visible satellites, their relative locations and possibly other criteria. The result is usually provided as uncertainties in horizontal (σ_{h}) and vertical (σ_{v}) direction. The exact meaning of those can differ from device to device but they can be treated as a standard deviation of the zero-mean Gaussian noise. This means that the measurement noise vector $\mathbf{n}_k^{(\text{GPS})}$ is expressed in the ENU frame, leading to the equation

$$\mathbf{r}_k^{(\text{GPS})} = \mathbf{H}_k^{(\text{GPS})}\tilde{\mathbf{X}}_k + \mathbf{A}\left(\hat{\phi},\hat{\lambda}\right)\mathbf{n}_k^{(\text{GPS})} \tag{5.30}$$

Because $\hat{\phi}$ and $\hat{\lambda}$ are dependent on the filter state, the noise parameter does not satisfy the requirements of the EKF and cannot be used directly for the update. The whole equation needs to be modified first

$$\mathbf{A}^T\left(\hat{\phi},\hat{\lambda}\right)\mathbf{r}_k^{(\text{GPS})} = \mathbf{A}^T\left(\hat{\phi},\hat{\lambda}\right)\mathbf{H}_k^{(\text{GPS})}\tilde{\mathbf{X}}_k + \mathbf{n}_k^{(\text{GPS})} \tag{5.31}$$

$$\mathbf{r}_k^{(\text{GPS,o})} = \mathbf{H}_k^{(\text{GPS,o})}\tilde{\mathbf{X}}_k + \mathbf{n}_k^{(\text{GPS,o})} \tag{5.32}$$

The measurement model in this form fulfills all the requirements of the EKF and can be used to update the filter state. The vector $\mathbf{n}_k^{(\text{GPS,o})}$ has a covariance matrix $\mathbf{R}^{(\text{GPS,o})}$

$$\mathbf{R}^{(\text{GPS,o})} = \begin{bmatrix} \sigma_{\text{h}}^2 & 0 & 0 \\ 0 & \sigma_{\text{h}}^2 & 0 \\ 0 & 0 & \sigma_{\text{v}}^2 \end{bmatrix} \tag{5.33}$$

5.5.4 Constraints fusion

Constraints formed by the residual $\mathbf{r}_k^{(\mathrm{GPS,o})}$ and matrices $\mathbf{H}_k^{(\mathrm{GPS,o})}$ and $\mathbf{R}^{(\mathrm{GPS,o})}$ can now be fused with the constraints from the camera from eq. (4.74). The fused measurement model is

$$\mathbf{r}_o \leftarrow \begin{bmatrix} \mathbf{r}_o \\ \mathbf{r}_k^{(\mathrm{GPS,o})} \end{bmatrix} \qquad \mathbf{H}_\mathbf{X} \leftarrow \begin{bmatrix} \mathbf{H}_\mathbf{X} \\ \mathbf{H}_k^{(\mathrm{GPS,o})} \end{bmatrix} \qquad \mathbf{n}_o \leftarrow \begin{bmatrix} \mathbf{n}_o \\ \mathbf{n}_k^{(\mathrm{GPS,o})} \end{bmatrix} \tag{5.34}$$

The filter can then proceed normally from the marginalization step described in (4.6.4) on page 42.

5.6 Summary

This chapter focused on one of the primary contributions of this work - a fusion of the GPS with the MSCKF.

It discussed the theoretical background of the GPS measurements, or more generally GNSS measurements, and the problematic of fusing such a measurement with the MSCKF.

At first, it described the basic principle of GNSS and GPS in particular, how the measurements are formed and which factors influence the accuracy. A few basic geodetic coordinate frames were described, including the conversions between them. Next, multiple options how to define the residuum of the measurement for the MSCKF were proposed. The residuum in ECEF frame was the most suitable of all the options and was then used to derive the measurements model, including the Jacobians involved. Finally, it was shown how such a measurement model can be used with the MSCKF itself.

6. Implementation details

This chapter discusses some important considerations of the implementation used for the experiments.

6.1 Description

The MSCKF was implemented as described in chapter 4. It was implemented in Python running on a MacBook Pro Min 2014 model with 2.8 GHz Intel Core i5 CPU. For some of the computer vision parts, OpenCV [67] was used. All of the data was processed offline.

The filter runs generally in about halt the speed of the real time. Roughly two thirds of the time is spent on image processing with the remaining time begin mostly the EKF steps itself. Both parts can be significantly speeded up by optimizing the code. The Python interpreter itself has a significant overhead too but the real-time implementation was not a goal and it was already proven by other authors that the filter is capable of the real time performance. The potential implementation on the device is left as a future work.

For source code, see the appendix B.

6.2 Feature tracking

The feature tracking subsystem uses the FAST feature detector, mainly for its fast extraction times. For the feature tracking itself, KLT tracker was used. Both of these were described earlier in section 3.3.2 and their implementation from OpenCV library [39] was used.

The FAST implementation used, extracts a predefined number of features with the highest response (a metric of being a feature). This has one, not so obvious, fail case. When there is a region in the image with rich texture, all the detected keypoints tend to be from that region, leaving out remaining parts of the image. It leads to poorly conditioned EKF update step. To enforce the uniformness of feature distribution, an image was split into 4×5 grid and features were sampled independently from each part.

Because of the iterative nature of the KLT tracker, the bigger the movement is (movement of a point in pixel coordinates), the more iterations it takes for the tracker to converge. By utilizing an extra information from the filter, a homography matrix between each two consecutive images was estimated. It was then used to provide a better guess of the feature position in the next frame, lowering the number of iterations required.

If the former image was captured by camera $\{C_i\}$ and the later one by camera

$\{C_{i+1}\}$[1], the relative rotation between the two is

$$
\begin{aligned}
{}^{C_{i+1}}_{C_i}\mathbf{R} &= {}^{C_i}_{G}\mathbf{R}\,{}^{G}_{C_{i+1}}\mathbf{R} && (6.1)\\
&= \left({}^{C}_{B}\mathbf{R}\,{}^{B_i}_{G}\mathbf{R}\right)\left({}^{C}_{B}\mathbf{R}\,{}^{B_{i+1}}_{G}\mathbf{R}\right)^T && (6.2)\\
&= {}^{C}_{B}\mathbf{R}\,{}^{B_i}_{G}\mathbf{R}\,{}^{B_{i+1}}_{G}\mathbf{R}^T\,{}^{C}_{B}\mathbf{R}^T && (6.3)
\end{aligned}
$$

The rotation matrices ${}^{C}_{B}\mathbf{R}$ and ${}^{B_i}_{G}\mathbf{R}$ are already (at the time of feature extraction as in the Algorithm 4) available in the filter state but matrix ${}^{B_{i+1}}_{G}\mathbf{R}$ is not. By reordering the steps of the algorithm and executing the propagation step before the feature tracking step, the value for ${}^{B_{i+1}}_{G}\mathbf{R}$ can be used after the prediction step. Because the interval between two consecutive images is in general small, the estimate from propagation is accurate enough.

The homography between two images depends on their rotation and on the camera matrix $\mathbf{K}$

$$
\begin{aligned}
\mathbf{H} &= \mathbf{K}\,{}^{C_{i+1}}_{C_i}\mathbf{R}\,\mathbf{K}^T && (6.4)\\
\mathbf{K} &= \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} && (6.5)
\end{aligned}
$$

Using the homography matrix $\mathbf{H}$, a feature position can be predicted as

$$
\begin{bmatrix} au_{i+1} & av_{i+1} & a \end{bmatrix}^T = \mathbf{K}\begin{bmatrix} u_i & v_i & 1 \end{bmatrix}^T \tag{6.6}
$$

where (u_i, v_i) are pixel coordinates of the feature in the first image and (u_{i+1}, v_{i+1}) are estimated pixel coordinates of the same feature in the next image.

The estimate is then fed to the KLT tracker as an initial guess, speeding up the computation.

Since the tracker can output false feature matches, a two point RANSAC algorithm, estimating the essential matrix, was used to filter out the outliers. It was based on the work from other authors [57, 68].

6.3 Pose pruning

In all the experiments, it was observed that the pruning schema (described in section 4.6.7) has a large effect on the quality of the result. A schema or pruning of the poses $\boldsymbol{\pi}_{B_2}, \boldsymbol{\pi}_{B_4}, \boldsymbol{\pi}_{B_7}$ was used.

6.4 Intensity of the gravitational acceleration

In the eq. (4.34) the exact gravitational acceleration at a given location on Earth is needed. It can be determined by the database lookup [69] or it can be measured by the accelerometer when the device stands still. In this work, the second

[1]The term camera $\{C_i\}$ means a camera at the time t_i. This shorthand is commonly used in multi-view geometry as the same camera at different time or multiple cameras at the same time are effectively the same thing.

approach was used. It is assumed that the device is static for first two seconds at the beginning of each dataset and the gravitational acceleration is estimated by calculating the average length of all the accelerometer measurements.

All the datasets were collected such that this condition holds.

7. Experimental evaluation

In this chapter, an evaluation of the proposed GPS fusion algorithm is performed.

The implementation of the vanilla MSCKF is first validated on a publicly available dataset with known ground truth and compared to results reported by other authors. Because this does not have GPS data, an evaluation is also done on a custom dataset, collected by a smartphone. This dataset is then used for evaluation of the proposed GPS fusion algorithm and its robustness.

7.1 Evaluation on public dataset

These experiments seek to verify the correctness of the implementation by running the filter on a publicly available dataset and comparing it to the performance characteristics reported by other authors.

A comparison of the MSCKF with other methods in not a goal of this thesis and was not performed, as it was already done by others [57, 30].

7.1.1 The dataset

The EuRoC dataset [70] was chosen for two major reasons. First, it contains accurate ground truth captured using the Vicon Motion capture system for the *Vicon room* sequences and by using accurate LIDAR scans for the *Machine hall* sequences. Second, data from the IMU and camera are properly synchronized by the hardware, ruling-out possible synchronization issues.

This dataset contains multiple sequences, captured by a flying quadrotor, equipped with, besides other sensors, a global shutter camera and an IMU.

7.1.2 Experiment description

The evaluation was performed on sequences *V2_02_medium* and *V2_03_difficult*. They were chosen because they contain very wide variety of motion, including some very challenging situations with rapid motion, which leads to a failure of many other localization methods.

The filter was run on the whole sequences with the same calibration parameters for both of them.

7.1.3 Evaluation methodology

Ground truth and estimated trajectory were aligned together by their first poses. Anther commonly used option is to find the best suitable transformation between the two trajectories. An example of this method was proposed by Umeyama [71]. The latter approach leads to lower error numbers. To properly time-synchronize the ground truth, it was approximated using 5th order Bézier curve [72] and sampled after every update of the filter.

The position estimation error was measured by two different metrics. The first metric was the *percentage of the total traveled distance* (%TTD). It is a ratio between the latest position error and the total distance that the robot has

traveled so far. This metric was used as it was the primary metric used by the authors of the MSCKF. Because this metric only takes into account the latest position error, the *root mean square error* (RMSE) defined as

$$E_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left({}^{G}\mathbf{p}_{B_{\text{GT}}}(t_k) - {}^{G}\hat{\mathbf{p}}_{B}(t_k)\right)^2} \tag{7.1}$$

where ${}^{G}\mathbf{p}_{B_{\text{GT}}}(t_k)$ is the ground truth position in the global frame, at time t_k. The RMSE metric reflects the whole trajectory, was measured as well. This is a very standard metric, which can be easily interpreted. RMSE is a single number calculated over the whole dataset.

Both %TTD and RMSE metrics are measuring an error of the estimate and the lower numbers mean more accurate estimates.

7.1.4 Results

Fig. (7.1) shows the plots of %TTD error and RMSE for both sequences.

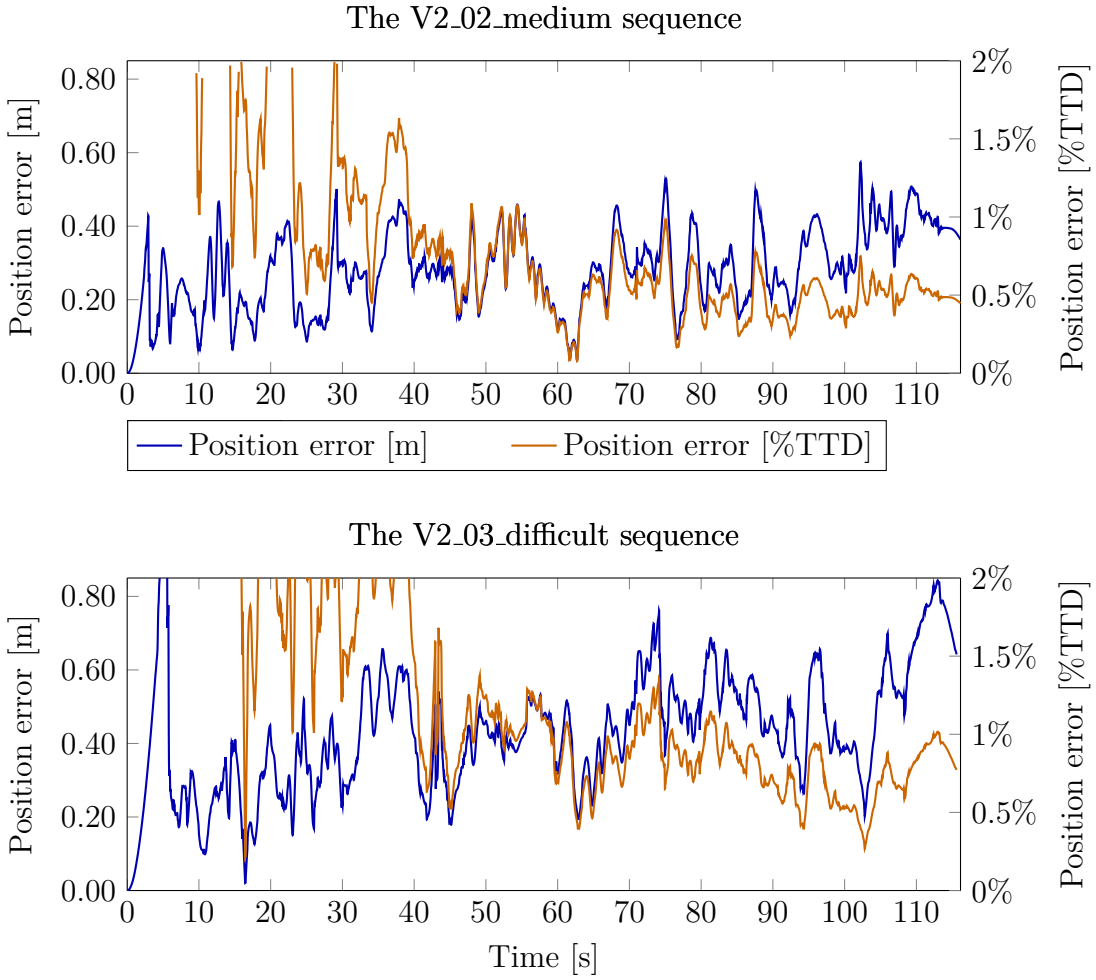


Figure 7.1: Position estimate errors for EuRoC dataset.

It can be seen that even though the sequences are very challenging, the maximum position error never exceeded 0.6 m over 81.16 meters traveled for the

V2_02_medium and 0.8 m over 83.16 meters traveled for the V2_03_difficult sequence. This corresponds to 0.44 %TTD and 0.301 m RMSE for the first sequence and and 0.65 %TTD and 0.45 m RMSE for the second sequence. As the filter does not utilize any mechanism for the global drift reduction, the error would drift indefinitely. This phenomenon is almost unnoticeable in the first sequence and very slow for the second one.

One observation is that the %TTD error metric is very high at the beginning. This is caused two facts. First, the traveled distance at the beginning is very small, which amplifies even small errors. Second, at the beginning, some variables might not be estimated by the filter as accurately as they will be later, making the position estimate "jump" a little. Later in the sequence, this metric tends to converge towards some small region. On both datasets, the %TTD error was getting lower during the whole sequence. If the dataset continued, it would possibly reach even lower error.

By closer examination of the error it can be noticed that despite being relatively low, it is not very smooth. One could observe small oscillations of the estimated position around the ground truth position. One possible cause of this behavior is that the calibration parameters are suboptimal, which might lead to not as accurate evaluation of the Jacobians and therefore inability to converge to a stable region. Another reason could be linearization errors. This kind of errors in inevitable for the used estimator.

Mourikis and Roumeliotis [19] reported an error on a custom dataset of 0.31 %TTD. Sun et al. [57] reported an error of roughly 0.3 m RMSE[1] for the V2_02_medium dataset and their approach diverged on the V2_03_difficult sequence. The results of this work are even more interesting as Sun et al. used stereo camera and this work only used a monocular camera. This might indicate that the second camera only provides a limited added value to the filter, at least in the situations captured in this particular sequence.

The comparison with results from other authors shows that the implementation performs well, even similarly to a stereo-camera based approach. The estimation accuracy is consistent with the results from previous research.

7.2 Evaluation on the smartphone

The evaluation was also performed on a custom dataset. The data was collected using an off-the-shelf iPhone 7 Plus.

The goal of this experiment was to assess the estimation accuracy of the filter when running with the low-grade sensors.

7.2.1 Issues

There are many significant differences from the EuRoC dataset used in the first experiment. A few of the most significant ones are discussed here.

First, the camera and the IMU cannot be synchronized by the hardware. This leads to the necessity of proper online calibration of the time offset t_d. It was

[1] Exact numbers were not provided and the estimated number was approximated from the graph in the article.

noticed during the experiments that even synchronization issues as small as 10 ms can lead to rapid divergence of the estimator.

The second issue is lack of the information about the hardware and software used to deliver the measurements. For example, each measurement is delivered with a timestamp but there is no guarantee of what this timestamp represents and which clock assigned it. An unexpected behavior of timestamps was experienced many times during the experiments and it commonly led to filter divergence.

The third issue is that an iPhone does not allow a developer to set the focal length of the camera to a known value. This can cause the camera using different focal length for each dataset, which limits the possibility to accurately calibrate the camera.

The fourth problem, also reported by Shelley [62], is the temperature. All the components of an iPhone are miniaturized and tightly packed into a small space. This causes the heat to be transferred unevenly from the CPU to the IMU, leading to a change in its noise characteristic. As there is not thermal sensor available on the IMU, this cannot be calibrated.

7.2.2 Calibration

To assure the correct behavior of the filter, some of the quantities needed to be accurately estimated in advance.

Some of the parameters change slowly in time. For example the IMU bias changes with temperature, t_d might be different every time the app for data collection is restarted and others. This practically limits the ability to accurately calibrate all the parameters. For those time-dependent parameters which can be estimated online by the filter, the best guess was used and the corresponding initial values of the covariance matrix were set to large values (the exact values can be found on the attached DVD, described in appendix B). The estimate then converges as the values become observable for the filter. For this reason, each dataset starts with a short sequence of random motion, exciting all axis of motion. This helps to speed up the convergence and avoids local minima.

IMU calibration

The IMU noise parameters $\boldsymbol{n}_g^T, \boldsymbol{n}_{wg}^T, \boldsymbol{n}_a^T, \boldsymbol{n}_{wa}^T$ were calibrated using the Allan variance [73] but the values obtained did not provide reliable results. The erroneous values might have been caused by the IMU not being perfectly still during the experiment, possibly because of relatively heavy traffic nearby. In the end, these values together with σ_{im} were found using a genetic algorithm optimization. Because the values were found so that the filter would work well with them, they likely contain some bias.

The parameters $\mathbf{b}_g$ and $\mathbf{b}_a$ were both set to zero with high initial uncertainty.

Camera and camera-to-IMU calibration

All the parameters of the camera model, namely $f_x, f_y, o_x, o_y, k_1, k_2, k_3, t_1, t_2$ and the camera-to-IMU temporal calibration t_d and spatial calibration ${}^B_C\mathbf{R}, {}^B\mathbf{p}_C$ were calibrated using Kalibr [61].

For the purpose of calibration, a specialized dataset was collected as required by Kalibr. It was important to collect the data by using the same settings of the phone, as used for the experiments.

The camera-to-IMU spatial and temporal calibration parameters only need to be estimated roughly as the filter estimates them in the runtime. The rest of the parameters need to be estimated as accurately as possible. Inaccurate parameters often lead to either immediate divergence of the filter or even to divergence after a longer period of time. The divergence is typically presented as a rapid change in the position estimate. Even when the filter does not diverge, inaccurate calibration results in degraded estimation performance.

7.2.3 IMU Interpolation

Some IMUs can sample the data from the gyroscope and the accelerometer at the same time. Because the iPhone used in the experiments is not capable of that and samples both sensors independently, they first needed to be synchronized.

It was observed that the gyroscope is sampled at a slightly higher frequency. To avoid loss of this extra bit of information, the gyroscope output was treated as a fixed sequence and the accelerometer measurements were calculated using linear interpolation for each gyroscope measurement. This synchronization was performed offline in batch processing.

Also, because the camera provides measurements independently of the IMU, the IMU measurements needed to be interpolated into the estimated time of the image capture $t + t_d$. Since t_d varies during each run, this was done in run time.

7.2.4 Dataset

The dataset used in this experiment consists of two sequences. Both of them were collected when walking with a handheld device.

One of the sequences was recorded in a park in a city. The other one was recorded in a residential area of a village. They are called *park* and *residential* respectively.

The images for these experiments were recorded at 15 Hz, with resolution of 640×480. The IMU was recording at the highest possible sample rate of roughly 100 Hz.

A dataset from a driving car was also recorded but was not used for the experiments. The vibrations from the engine caused severe rolling shutter artifacts which in terms caused the filter to diverge quickly. The problem can be solved by a better hardware realization of the experiment.

7.2.5 Evaluation methodology

Since there is no ground truth available for this dataset, it is not possible to use the same evaluation method as in the case of the EuRoC dataset.

The evaluation was performed in a similar way to other work in this area i.e. [19]. The estimation accuracy was assessed by aligning the beginning of the resulting trajectory with the photo of the area from the satellite and measuring the final translation error. The traveled distance was estimated by measuring the estimated ground truth trajectory on the map.

This evaluation method is only approximate. The goal of this experiment is to provide baseline results for the following experiments with the GPS which will be evaluated in the same way. Therefore, limited evaluation accuracy should not be an issue.

7.2.6 Results

In fig. (7.2), both trajectories are projected onto a map together with the approximated ground truth.

The estimation accuracy for both sequences is summarized in table (7.3).

sequence	final error [%TTD]	final error [m]	trajectory length
park	4.92	9.77	198.47
residential	8.76	27.19	310.3

Figure 7.3: Summary of the experimental results on the smartphone.

7.2.7 Discussion

It can be noticed that the estimation accuracy is considerably lower comparing to the EuRoC dataset.

The most likely explanation is the insufficiently accurate model of the IMU noise parameters. The possible solution is to use the more sophisticated models (3.3) and (3.6). These were used by Li et al. [24], where they reported the final errors of only 0.17 %TTD using a similar smartphone. This work differs in few other key components, so a direct conclusion cannot be drawn but the authors pointed that the accurate sensor modeling is the main reason behind those low numbers. Another plausible explanation might be inaccurate calibration. A further research of this issue is considered a future work.

The relatively poor performance of the filter is not an obstacle for further experiments. In fact, it will help to amplify the added value of the fusion with the GPS.

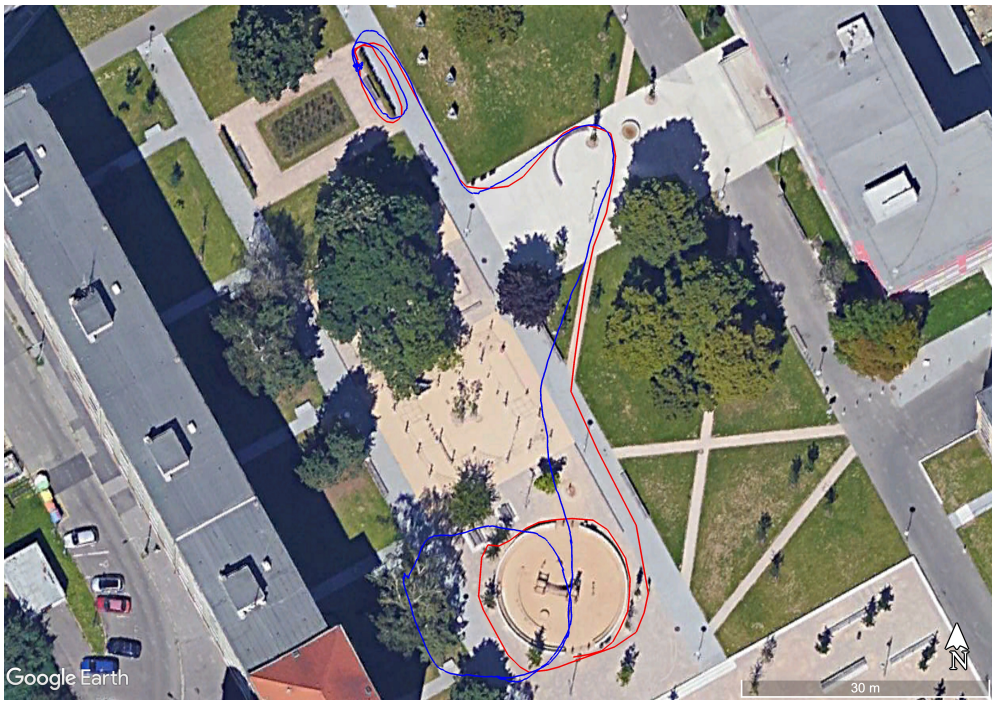
7.3 Evaluation of the GPS fusion algorithm

This experiment focuses on the proposed fusion of the MSCKF with the GPS, as described in chapter 5.

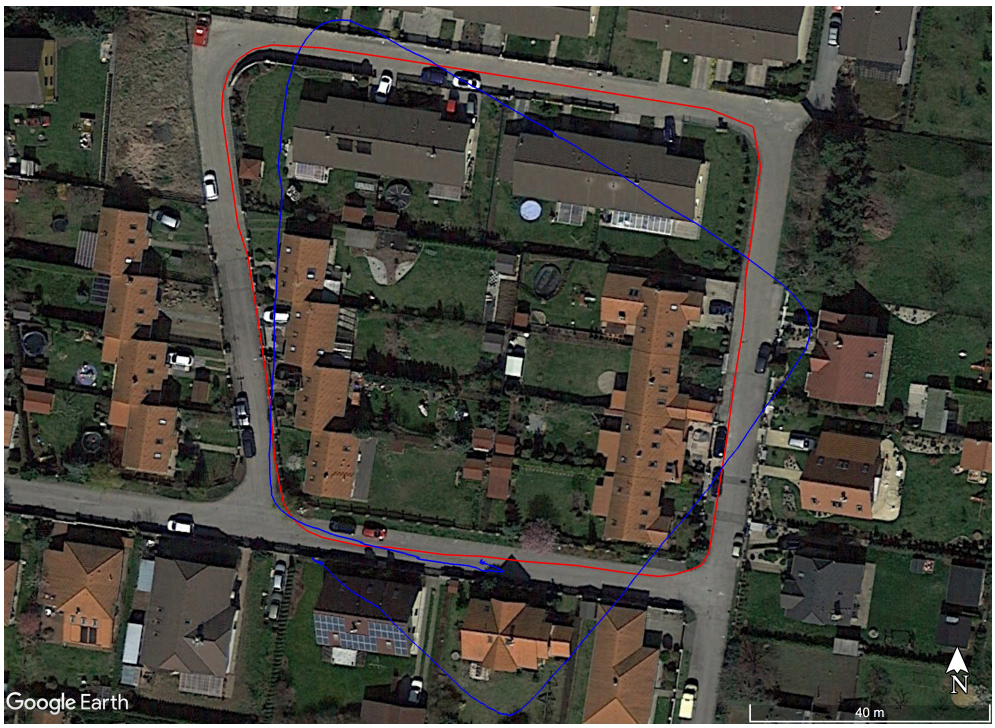
For this experiment, the same dataset and evaluation methodology will be used. This way the direct consequence of the fusion can be observed ruling out the possible external influences which might skew the results.

Fig. (7.4) visualizes all the GPS measurements recorded. It can be seen that the measurements roughly approximate the trajectory but they contain a significant amount of noise. The park sequence is more affected by this issue than the residential one.

To assure fairness of the comparison, the initial guess of the parameters ϕ_0, λ_0, h_0 and ψ (as described in chapter 5) was chosen such that it represents the same alignment as in the previous experiment.

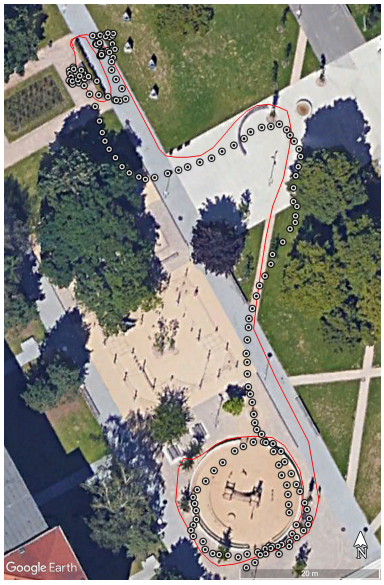


(a) The park sequence



(b) The residential sequence

Figure 7.2: The result of the experiment on an iPhone, without GPS. The blue line is the trajectory estimated by the filter and the red line is the approximated ground truth.



(a) The park sequence

(b) The residential sequence

Figure 7.4: Visualization of the GPS measurements. Each dot represents a single measurement. The red line is the approximated ground truth.

7.3.1 Results

Because of the high level of noise in the park sequence, the uncertainty levels reported from the GPS had to be increased by a factor of 3. Without this modification the parameters prematurely converged to incorrect values. The reason is the high noise at the beginning of the sequence, as discussed above.

The results of the experiment are again superimposed over the map of the area. The result is depicted in fig. (7.5). For easier comparison, results of the previous experiment are shown as well.

The numerical evaluation of the results is summarized in table (7.6).

sequence	final error [%TTD]	final error [m]	trajectory length
park	1.7	3.39	198.47
residential	0.89	2.78	310.3

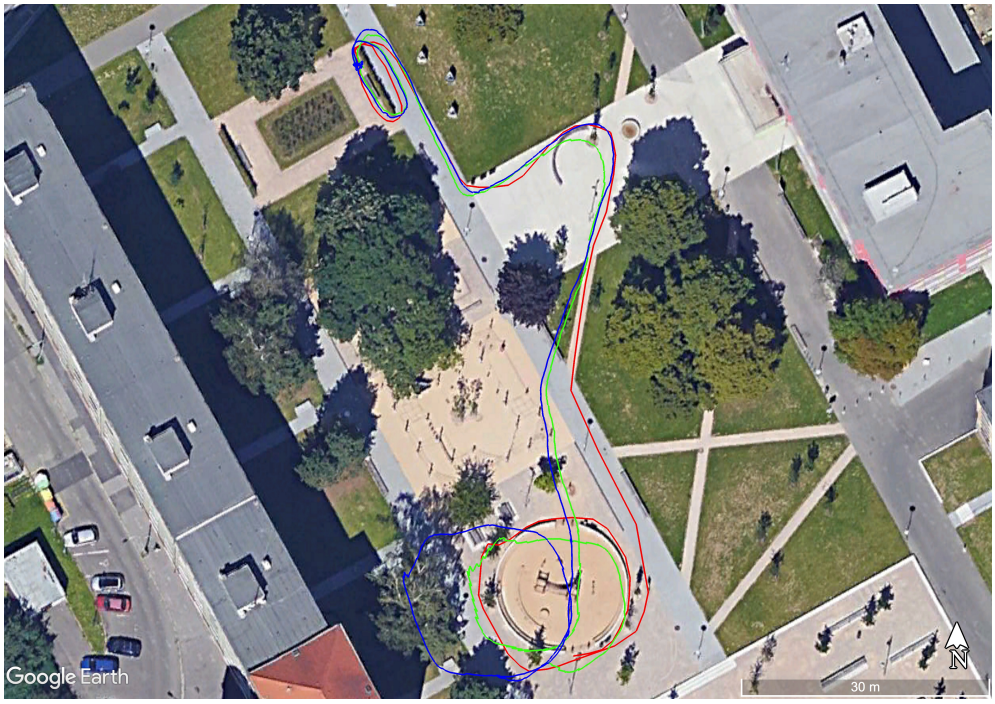
Figure 7.6: Summary of the experimental results without the GPS.

7.3.2 Discussion

It can be easily seen that the proposed GPS fusion algorithm has a significant impact on the result.

In case of the residential sequence, the estimated trajectory follows the ground truth very precisely, which resulted in the final error only twice as large as for the V2_02_medium sequence of the EuRoC dataset and only 37% higher then the error for the V2_03_difficult sequence. Especially when considering the fact that the EuRoC dataset was recorded using expensive and specialized hardware, the results are promising.

The park sequence showed an interesting behavior. First, note that this sequence is considerably more challenging than the residential one. The GPS mea-



(a) Park sequence

(b) Residential sequence

Figure 7.5: Results of the experiment fusing the filter with the GPS. The estimated trajectory during this experiment is highlighted by a green color. The blue and red lines represent the results without the GPS and the ground truth respectively.

surements were less accurate during the whole sequence but the most difficult part is the beginning. By close inspection of the initial part of the GPS readouts in fig. (7.4a), it can be seen that the sensor consistently reports strongly biased numbers. In this initial phase, the filter is still in the phase of the graduate convergence of the estimated variables. This makes it more sensitive to outliers and inaccurate data which was experienced many times during the experiments. Both facts combined led to the slower convergence of the GPS-related parameters and ψ in particular. In the meantime, the position estimate drifted further away from the true trajectory. The accumulated error was then corrected near the end of the sequence.

One can also observe that when compared with the previous experiment, the trajectory is less smooth. There are two factors which caused this effect. The first is the fast position drift of the bare filter and the second is low rate of the GPS measurements. A combination of those caused a large position drift in between two GPS measurements.

It was necessary to artificially inflate the uncertainty of the GPS measurements for the park sequence due to its high noise. This step should not be needed or at least it should not need to be inflated by such a large amount. The reported uncertainties of 6 to 8 meters are approximately accurate. The possible cause might be a smoothing performed by the sensor itself. Some sensors do not treat each measurement independently and they apply some sort of smoothing to the consequent measurement. This then violates the assumption of the independence of the noise of each measurement, as described in 5.5.3. A further experimentation is needed to better understand the true noise characteristics of the GPS sensor provided in the iPhone. This problem is potentially only relevant for some devices.

This experiment showed the benefits of the GPS measurements for the visual-inertial odometry. In terms of accuracy, the proposed algorithm outperformed both GPS and MSCKF on its own.

7.4 GPS outage experiment

To test the robustness of the proposed algorithm to GPS outages, another experiment was conducted. It is important for the algorithm to be able to deal with a GPS outage as it is very likely to happen for example when a car goes into a tunnel or when a robot operates partially indoors and partially outdoors.

7.4.1 The dataset

The experiment was conducted on the residential sequence with a simulated GPS outage. The park sequence was not used because the estimate of the position and orientation in the world converged very close to the end of the sequence. With a simulated outage, those quantities would likely not even converge by the end of the sequence.

The area with the simulated outage is outlined by a light blue color in fig. (7.7). The robot was inside of the affected area for about one minute, which is roughly a quarter of the whole sequence.

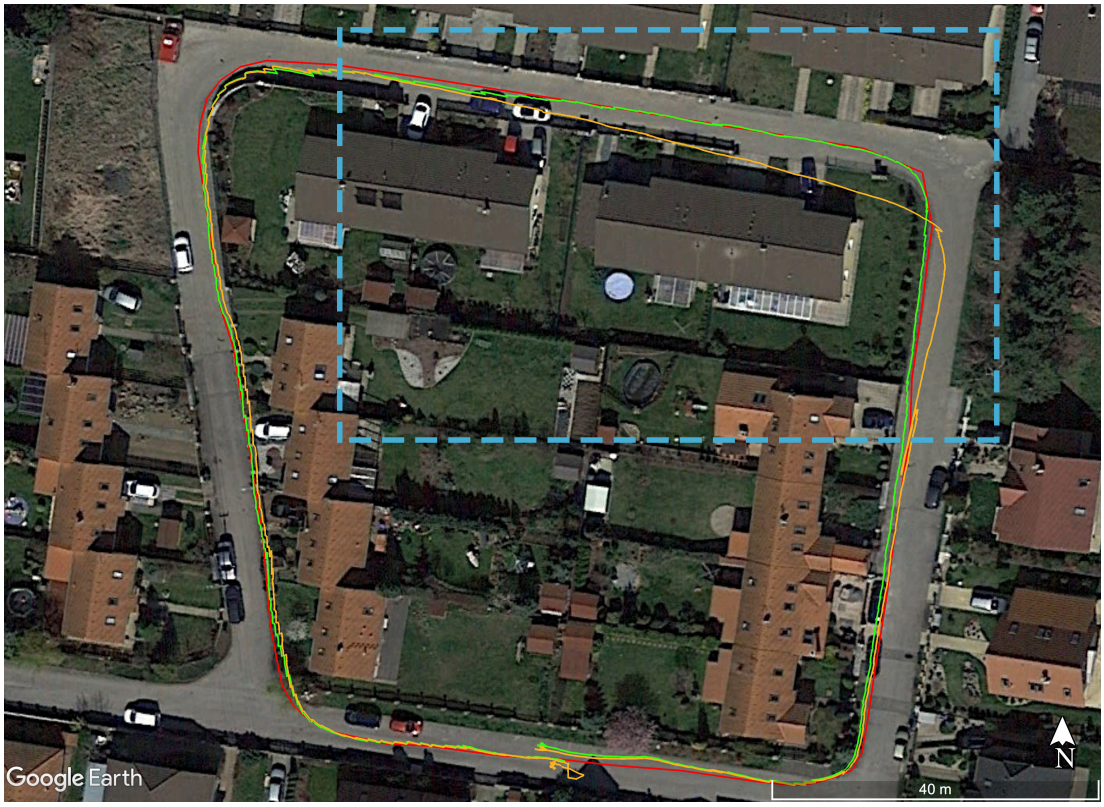


Figure 7.7: The results of the GPS outage experiment on the residential sequence. Orange color represents the trajectory from this experiment. For comparison, the result of the experiment without the outage is included in green color and the ground truth is outlined in red.

7.4.2 Results

The trajectory estimated by the filter is depicted by orange line in fig. (7.7) and the final error is summarized in table 7.8.

sequence	final error [%TTD]	final error [m]	trajectory length
residential	0.68	2.12	310.3

Figure 7.8: Summary of the GPS outage experiment.

7.4.3 Discussion

It can be seen that the position estimate slowly degraded as the corrective measurements were not available. Once the outage period ended, the filter recovered most of the drift with a single GPS measurement.

The final position error was slightly lower than the error produced on the same sequence without the simulated outage but this is likely just circumstantial and might not appear for other datasets.

In a real-world situation, a GPS outage would likely be accompanied by a few inaccurate measurements both before and after the outage. If the sensor would correctly report high uncertainties, this should not influence the estimation accuracy. This speculation was not tested and is based purely on the overall results, mainly the high-noise experiment in section 7.5.

7.5 High measurement noise experiment

As the raw GPS readouts can be sometimes heavily encumbered with a noise, an experiment testing the behavior under such conditions was performed. The goal was to see whether the proposed algorithm would be able to deal with noise and what will the behavior look like. The experiment will test whether the filter would be able to process the noisy measurements or whether the extra noise will cause it to diverge. In addition, it will examine the loss of accuracy in comparison to using GPS measurements with no added noise and utilizing no GPS measurements at all.

The filter was not fed any GPS measurements during the initial convergence phase. This does not bias the experiment as it can be done regardless to the added noise. The filter was then only fed with high noise measurements.

All initial values were set to the same numbers as in the previous experiments.

7.5.1 The dataset

The experiment was carried out on the residential sequence. All the sensor measurements but GPS were left unchanged. The extra noise was added on top of the raw GPS measurements obtained from the device. The added noise was Gaussian, independent noise in horizontal plane with a standard deviation of 30 meters. The measurement noise uncertainty was adjusted accordingly.

7.5.2 Results

As this experiment is highly randomized by its nature, it was repeated 10 times. The errors for all individual trials are summarized in table 7.9.

trial	final error [%TTD]	final error [m]
1	4.62	14.34
2	4.57	14.17
3	3.81	11.84
4	5.29	16.44
5	4.42	13.72
6	5.82	18.07
7	3.56	11.05
8	10.53	32.70
9	3.59	11.13
10	6.95	21.58
Average	**5.32**	**15.50**

Figure 7.9: Summary of the GPS outage experiment.

Fig. (7.10) shows the GPS measurements from the first trial and estimated trajectories from all 10 trials.

7.5.3 Discussion

The noise levels in this experiment (30 meters in horizontal plane) were chosen such that they roughly reflect the expected worst-case scenario. In a real-world

(a) Noisy measurements from one of the trials.

(b) Trajectories from all of the trials.

Figure 7.10: The data and results from the experiment with high GPSmeasurement noise. Each run is represented by a single white line in the bottom image. In both images, the red line represents the approximated ground truth.

scenario, it can be expected that at least some measurements will be more accurate. The amount of noise is so high that no obvious path is visible by observing the measurements in fig. (7.10a).

The influence of the noise can be seen on all the trials of the experiment. This is expected as the Kalman filter does not perform any smoothing. It moves the state towards the predicted state given by the measurement while the distance of the movement is given by the Kalman gain. All the trajectories are also skewed towards the center of the loop.

Despite that, the filter was able to extract some information from the measurements. The average %TTD error was 5.32 whereas the same trajectory without utilizing any GPS information had an error of 8.76 %TTD.

In none of the trials did the filter diverge and it was able to track the robot until the end of the sequence. This property would be especially beneficial for applications like self-driving cars where the filter should work under all circumstances. If the filter would diverge in the presence of high measurement noise, it might draw this approach unusable for such application.

7.6 Summary

This chapter focused on the experimental evaluation of the proposed algorithm and its properties under different condition.

It was shown that the implementation of the MSCKF is correct and performs comparably to similar approach utilizing stereo camera.

A series of experiments showed that the proposed algorithm is not only capable of tracking the robot but was able to outperform both GPS and MSCKF separately. In experiments conducted, the improvement was by an order of magnitude.

The experiments focused on the robustness under different conditions showed the ability to operate with highly corrupted measurements by a noise, while still improving the accuracy, or even in presence of substantial GPS outages.

Conclusion

This work was focusing on the problematics of lifelong localization of robots without an assumption of a restricted operational area. The selected approach started with the MSCKF algorithm and fused it with GPS measurements in order to keep the position estimate drift bounded in time.

The MSCKF was described in a great detail and the algorithm to fuse the GPS measurements was presented. Experiments showed that in terms of accuracy, the proposed technique outperforms both GPS and the MSCKF alone. Experiments further demonstrate that the algorithm is able to reliably operate when the GPS signal is highly corrupted by noise or even in presence of substantial GPS outages.

7.7 Contributions

The technique presented in this work is to the best of our knowledge the first tight integration of the GPS and visual-inertial odometry described in literature. We believe that this combination of sensors has very powerful properties and will be further studied in the future.

Another contribution of this work is that it provides a public implementation. This implementation of MSCKF is the only one publicly available that is capable of working with low-grade sensors. The availability of an implementation can help to accelerate the MSCKF-related research and possibly even a further research of the proposed method.

Last but not least thing to emphasize is the detailed description of the MSCKF provided in this work. Current entry-barrier for researchers and users is quite high, partially due to a lack of detailed derivation of the method itself.

7.8 Future work

As a future work, the implementation of the filter should be optimized so that it is able to operate in real-time on a mobile device.

Also, implementation of more sophisticated camera and IMU models would presumably improve the accuracy when operating without the GPS.

A further experimentation is needed to better understand the noise characteristics of the GPS sensor used in the experiments. This can help to improve the accuracy of the filter even further, especially in the important environments with poor GPS reception such as urban valleys.

The proposed technique can be modified to operate with other positioning sensors instead of, or in addition to the GPS. For example, a robot can detect when it enters a known building and supplement the GPS measurements by using a pre-built map. This would open even broader spectrum of applications.

Bibliography

[1] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[2] Apple arkit. `https://developer.apple.com/arkit/`. Accessed: 2018-08-12.

[3] Google arcore. `https://developers.google.com/ar/discover/`. Accessed: 2018-08-12.

[4] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[5] Ryan W Wolcott and Ryan M Eustice. Visual localization within lidar maps for automated urban driving. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 176–183. IEEE, 2014.

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[7] R Schrijver. Precision agriculture and the future of farming in europe. *Scientific Foresight Study*, 2016.

[8] Amazon prime air. `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`. Accessed: 2018-07-13.

[9] Kelen CT Vivaldini, Jorge PM Galdames, Thales S Bueno, Roberto C Araújo, Rafael M Sobral, Marcelo Becker, and Glauco AP Caurin. Robotic forklifts for intelligent warehouses: Routing, path planning, and autolocalization. In *Industrial Technology (ICIT), 2010 IEEE International Conference on*, pages 1463–1468. IEEE, 2010.

[10] Masatoshi Arikawa, Shin'ichi Konomi, and Keisuke Ohnishi. Navitime: Supporting pedestrian navigation in the real world. *IEEE Pervasive Computing*, 6(3), 2007.

[11] Elias Mueggler, Matthias Faessler, Flavio Fontana, and Davide Scaramuzza. Aerial-guided navigation of a ground robot among movable obstacles. In *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pages 1–8. IEEE, 2014.

[12] Nataly Levesque and Harold Boeck. Proximity marketing as an enabler of mass customization and personalization in a customer service experience. In *Managing Complexity*, pages 405–420. Springer, 2017.

[13] Jindřich Havlík and Ondřej Straka. Performance evaluation of iterated extended kalman filter with variable step-length. In *Journal of Physics: Conference Series*, volume 659, page 012022. IOP Publishing, 2015.

[14] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[15] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): part i the essential algorithms. 2006.

[16] M. Li and A. I. Mourikis. Consistency of ekf-based visual-inertial odometry. Technical report, Dept. of Electrical Engineering, University of California, Riverside, May 2012.

[17] Joel A Hesch, Dimitrios G Kottas, Sean L Bowman, and Stergios I Roumeliotis. Observability-constrained vision-aided inertial navigation. *University of Minnesota, Dept. of Comp. Sci. & Eng., MARS Lab, Tech. Rep*, 1:6, 2012.

[18] Guoquan P Huang, Anastasios I Mourikis, and Stergios I Roumeliotis. Observability-based rules for designing consistent ekf slam estimators. *The International Journal of Robotics Research*, 29(5):502–528, 2010.

[19] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572. IEEE, 2007.

[20] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. Technical report, Dept. of Computer Science and Engineering, University of Minnesota, 2006.

[21] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual–inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

[22] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis. A first-estimates Jacobian EKF for improving SLAM consistency. In *Proceedings of the International Symposium on Experimental Robotics*, Athens, Greece, July 2008.

[23] Mingyang Li, Byung Hyung Kim, and Anastasios I Mourikis. Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4712–4719. IEEE, 2013.

[24] Mingyang Li, Hongsheng Yu, Xing Zheng, and Anastasios I Mourikis. High-fidelity sensor modeling and self-calibration in vision-aided inertial navigation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 409–416. IEEE, 2014.

[25] M. Li and A. I. Mourikis. Online temporal calibration for camera-imu systems: Theory and algorithms. *International Journal of Robotics Research*, 33(7):947–964, June 2014.

[26] Trung Nguyen, George KI Mann, Andrew Vardy, and Raymond G Gosine. Likelihood-based iterated cubature multi-state-constraint kalman filter for visual inertial navigation system. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 4410–4415. IEEE, 2017.

[27] Martin Brossard, Silvere Bonnabel, and Axel Barrau. Unscented kalman filter on lie groups for visual inertial odometry. 2018.

[28] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 298–304. IEEE, 2015.

[29] Michael Bloesch, Michael Burri, Sammy Omari, Marco Hutter, and Roland Siegwart. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072, 2017.

[30] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[31] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *arXiv preprint arXiv:1708.03852*, 2017.

[32] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen. Autonomous aerial navigation using monocular visual-inertial fusion. 00:1–29, 2017.

[33] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[34] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[35] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[36] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[37] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[38] Review: Sony fdr-ax100/b: Rolling shutter. `http://blog.joemacirowski.com/archives/1461`. Accessed: 2018-08-12.

[39] OpenCV. *The OpenCV Reference Manual*, 3.4.1 edition, June 2018.

[40] Mars science laboratory: Curiosity rover: Multimedia. `https://mars.nasa.gov/msl/multimedia/raw/`. Accessed: 2018-06-20.

[41] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[42] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[43] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[44] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[45] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[46] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[47] Chris Sweeney, Torsten Sattler, Tobias Hollerer, Matthew Turk, and Marc Pollefeys. Optimizing the viewing graph for structure-from-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 801–809, 2015.

[48] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse depth parametrization for monocular slam. *IEEE transactions on robotics*, 24(5):932–945, 2008.

[49] Peter S Maybeck. Stochastic models, estimation, and control. 1979.

[50] José A Castellanos, José Neira, and Juan D Tardós. Limits to the consistency of ekf-based slam. *IFAC Proceedings Volumes*, 37(8):716–721, 2004.

[51] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot. Consistency of the ekf-slam algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3562–3568. IEEE, 2006.

[52] Mingyang Li and Anastasios I Mourikis. Improving the accuracy of ekf-based visual-inertial odometry. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 828–835. IEEE, 2012.

[53] Mingyang Li and Anastasios Mourikis. Optimization-based estimator design for vision-aided inertial navigation. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[54] Mingyang Li and Anastasios I Mourikis. Vision-aided inertial navigation for resource-constrained systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1057–1063. IEEE, 2012.

[55] Mingyang Li and Anastasios I Mourikis. 3-d motion estimation and online temporal calibration for camera-imu systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5709–5716. IEEE, 2013.

[56] Mingyang Li and Anastasios I Mourikis. Online temporal calibration for camera–imu systems: Theory and algorithms. *The International Journal of Robotics Research*, 33(7):947–964, 2014.

[57] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *CoRR*, abs/1712.00036, 2017.

[58] Xing Zheng, Zack Moratto, Mingyang Li, and Anastasios I Mourikis. Photometric patch-based visual-inertial odometry. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3264–3271. IEEE, 2017.

[59] Hongsheng Yu and Anastasios I Mourikis. Edge-based visual-inertial odometry. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 6670–6677. IEEE, 2017.

[60] Joan Solà. Quaternion kinematics for the error-state kalman filter. *CoRR*, abs/1711.02508, 2017.

[61] Joern Rehder, Janosch Nikolic, Thomas Schneider, Timo Hinzmann, and Roland Siegwart. Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4304–4311. IEEE, 2016.

[62] M. Shelley. Monocular visual inertial odometry on a mobile device. Master's thesis, Technical University Munich, Germany, Aug. 2014.

[63] Bradford W Parkinson, Per Enge, Penina Axelrad, and James J Spilker Jr. *Global positioning system: Theory and applications, Volume II*. American Institute of Aeronautics and Astronautics, 1996.

[64] GPS Directorate. Navstar gps space segment/user segment l5 interfaces (is-gps-705b). Technical report, Technical report, 2011.

[65] Jijie Zhu. Conversion of earth-centered earth-fixed coordinates to geodetic coordinates. *IEEE Transactions on Aerospace and Electronic Systems*, 30(3):957–961, 1994.

[66] Bernard Russel Bowring. Transformation from spatial to geographical coordinates. *Survey review*, 23(181):323–327, 1976.

[67] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[68] Bernd Kitt, Andreas Geiger, and Henning Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 486–492. IEEE, 2010.

[69] Nikolaos K Pavlis, Simon A Holmes, Steve C Kenyon, and John K Factor. The development and evaluation of the earth gravitational model 2008 (egm2008). *Journal of geophysical research: solid earth*, 117(B4), 2012.

[70] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[71] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):376–380, 1991.

[72] Colm Buckley. Bézier curves for camera motion. Technical report, Trinity College Dublin, Department of Computer Science, 1994.

[73] Naser El-Sheimy, Haiying Hou, and Xiaoji Niu. Analysis and modeling of inertial sensors using allan variance. *IEEE Transactions on instrumentation and measurement*, 57(1):140–149, 2008.

[74] Hipp, r, et. al., sqlite. `https://www.sqlite.org/download.html`. Accessed: 2018-08-12.

List of Abbreviations

EKF Extended Kalman Filter. 8, 27–29, 39, 54, 57

FOG fibre optic gyroscope. 14

GNSS global navigation satellite system. 47, 55

GPS Global Positioning System. 4, 5, 7, 11, 47, 48, 50–55, 61, 66, 68–75, 91, 92

IMU inertial measurement unit. 4, 6–8, 30–34, 36, 38, 43, 52, 53, 61, 63–66, 75

INS Inertial Navigation System. 27, 29

LIDAR Light Detection and Ranging. 7, 61

MEMS Microelectromechanical systems. 14

MSCKF Multi-State Constraint Kalman Filter. 8, 9, 11, 12, 31–33, 39, 47, 50, 53, 55, 57, 61, 62, 66, 70, 74, 75

RLG ring laser gyroscope. 14

RTK real-time kinematics. 7

SLAM Simultaneous Localization and Mapping. 5, 6, 8, 11

VIO visual-inertial odometry. 29, 47

VSG vibrating structure gyroscope. 14

A. Math

A.1 Rotations

Without any formal definition, we state that the *special-orthogonal group*, denoted as $SO(3)$, is the group of all rotations around origin, in $\mathbb{R}^3$.

Elements of this group can be represented in different ways. The ones we will be using are rotation matrices and unit-length quaternions.

The rotation matrices are orthogonal. Intuitively, with respect to the standard basis $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ of $\mathbb{R}^3$ the columns of $\mathbf{R}$ are given by $[\mathbf{Re}_1, \mathbf{Re}_2, \mathbf{Re}_3]$. Since the standard basis is orthonormal and $\mathbf{R}$ preserves angles and length, the columns of $\mathbf{R}$ form another orthonormal basis. This orthonormality condition can be expressed as

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}_3 \Rightarrow \mathbf{R}^{-1} = \mathbf{R}^T$$

In addition, for every $\mathbf{R} \in SO(3)$, $\det(\mathbf{R}) = 1$.

Definition A.1. Cross-product Let $\mathbf{u}, \mathbf{v} \in \mathbf{R}^3$, $\mathbf{u} = \begin{bmatrix} u_1, u_2, u_3 \end{bmatrix}^T$ and $\mathbf{u} = \begin{bmatrix} v_1, v_2, v_3 \end{bmatrix}^T$. The cross-product $\mathbf{u} \times \mathbf{v}$ is

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_1v_3 - u_3v_1 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

Lemma 1. *Properties of the cross-product For every* $\mathbf{R} \in SO(3)$ *and for every* $\mathbf{u}, \mathbf{v} \in \mathbf{R}^3$ *holds that*

$$\mathbf{R}(\mathbf{u} \times \mathbf{v}) = \mathbf{Ru} \times \mathbf{Rv}$$

Proof. Let $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ be the standard orthonormal basis of $\mathbb{R}^3$, $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^T$, $\mathbf{v} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^T$. Then

$$\mathbf{u} \times \mathbf{v} = \det \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix}$$

and for $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}^T$

$$\begin{aligned} (\mathbf{u} \times \mathbf{v})\,\mathbf{w}^T &= \det \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \left(w_1\mathbf{e}_1^T + w_2\mathbf{e}_2^T + w_3\mathbf{e}_3^T \right) \\ &= \det \begin{bmatrix} w_1 & w_2 & w_3 \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \\ &= \det \begin{bmatrix} \mathbf{w} & \mathbf{u} & \mathbf{v} \end{bmatrix}^T \end{aligned}$$

Taking the $\mathbf{R}$ as from lemma

$$\begin{aligned}(\mathbf{Ru}\times\mathbf{Rv})\,(\mathbf{Rw})^T &= \det\begin{bmatrix}\mathbf{Rw} & \mathbf{Ru} & \mathbf{Rv}\end{bmatrix}^T\\ &= \det\left(\begin{bmatrix}\mathbf{w} & \mathbf{u} & \mathbf{v}\end{bmatrix}^T\mathbf{R}^T\right)\\ &= \det\left(\begin{bmatrix}\mathbf{w} & \mathbf{u} & \mathbf{v}\end{bmatrix}^T\right)\\ &= \lfloor\mathbf{u}\times\mathbf{v}\rfloor\,\mathbf{w}^T\end{aligned}$$

Because

$$(\mathbf{Ru}\times\mathbf{Ru})\,(\mathbf{Rw})^T = \mathbf{R}^T\,(\mathbf{Ru}\times\mathbf{Ru})\,\mathbf{w}^T$$

it holds that

$$\begin{aligned}\mathbf{R}^T\,(\mathbf{Ru}\times\mathbf{Ru})\,\mathbf{w}^T &= (\mathbf{u}\times\mathbf{v})\,\mathbf{w}^T\\ \mathbf{R}^T\,(\mathbf{Ru}\times\mathbf{Ru}) &= (\mathbf{u}\times\mathbf{v})\\ \mathbf{Ru}\times\mathbf{Ru} &= \mathbf{R}\,(\mathbf{u}\times\mathbf{v})\end{aligned}$$

□

Definition A.2. $\lfloor\boldsymbol{\omega}_\times\rfloor$ is *cross-matrix* of the vector $\omega = \begin{bmatrix}\omega_x, \omega_y, \omega_z\end{bmatrix}^T$ defined as

$$\lfloor\boldsymbol{\omega}_\times\rfloor = \begin{bmatrix}0 & -\omega_z & \omega_y\\ \omega_z & 0 & -\omega_x\\ -\omega_y & \omega_x & 0\end{bmatrix}$$

Lemma 2. *Properties of the cross-matrix For every* $\mathbf{R}\in SO(3)$ *and for every* $\mathbf{u},\mathbf{v}\in\mathbf{R}^3$ *holds that*

$$\lfloor\mathbf{u}_\times\rfloor^T = -\lfloor\mathbf{u}_\times\rfloor \tag{A.1}$$

$$\lfloor\mathbf{u}_\times\rfloor\,\mathbf{v} = -\lfloor\mathbf{v}_\times\rfloor\,\mathbf{u} \tag{A.2}$$

$$\lfloor\mathbf{Ru}_\times\rfloor = \mathbf{R}\,\lfloor\mathbf{u}_\times\rfloor\,\mathbf{R}^{-1} \tag{A.3}$$

Proof. The equation (A.1) is proven by the fact that cross-matrix is skew-symmetric and for every skew-symmetric matrix $\mathbf{A}$ holds that $\mathbf{A}^T = -\mathbf{A}$.

Second equation (A.2) is proven trivially from the definition of the cross-matrix.

To prove the equation (A.3) let $\mathbf{R},\mathbf{u},\mathbf{v}$ be as in the lemma. Then it is true that

$$\begin{aligned}\mathbf{u}\times\mathbf{R}^{-1}\mathbf{v} &= \lfloor\mathbf{u}_\times\rfloor\,\mathbf{R}^{-1}\mathbf{v}\\ \mathbf{R}\left(\mathbf{u}\times\mathbf{R}^{-1}\mathbf{v}\right) &= \mathbf{R}\,\lfloor\mathbf{u}_\times\rfloor\,\mathbf{R}^{-1}\mathbf{v}\end{aligned}$$

Applying the lemma (1), the left side of the equation is

$$\mathbf{R}\left(\mathbf{u}\times\mathbf{R}^{-1}\mathbf{v}\right) = \mathbf{Ru}\times\mathbf{RR}^{-1}\mathbf{v} = \mathbf{Ru}\times\mathbf{v} = \lfloor\mathbf{Ru}_\times\rfloor\,\mathbf{v}$$

Combining this with the previous result leads to

$$\lfloor\mathbf{Ru}_\times\rfloor\,\mathbf{v} = \mathbf{R}\,\lfloor\mathbf{u}_\times\rfloor\,\mathbf{R}^{-1}\mathbf{v}$$

□

A.2 Quaternions

The quaternions are a generalization of the complex numbers into four dimensions, adding constants j and k on top of the i. It is a useful tool as they can represent a rotation in 3D without the drawbacks of other representations[1].

A quaternion is defined as

$$\mathbf{q} = iq_x + jq_y + kq_z + q_w \qquad \Leftrightarrow \qquad \mathbf{q} = \mathbf{q_v} + q_w \tag{A.4}$$

where

$$ji = -ij = k \qquad ik = -kj = i \qquad ki = -ik = j \tag{A.5}$$

It can be also represented as a vector

$$\mathbf{q} = \begin{bmatrix} \mathbf{q_v} \\ q_w \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_z \\ q_w \end{bmatrix} \tag{A.6}$$

The above formulas lead to the JPL convention of quaternions, which is used in our work. There are other alternatives, including the Hamilton notation. The differences between various conventions and more in-depth analysis of quaternions including proofs can be found in a very detailed tutorial by Solà [60].

Definition A.3. Product A product $\otimes$ of two quaternions $\mathbf{p}$ and $\mathbf{q}$ is

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} q_w p_x + q_z p_y - q_y p_z + q_x p_w \\ -q_z p_x + q_w p_y + q_x p_z + q_y p_w \\ q_y p_x - q_x p_y + q_w p_z + q_z p_w \\ -q_x p_x - q_y p_y - q_z p_z + q_w p_w \end{bmatrix} \tag{A.7}$$

Note that quaternion product is not commutative, meaning that generally

$$\mathbf{q} \otimes \mathbf{p} \neq \mathbf{p} \otimes \mathbf{q} \tag{A.8}$$

but it is associative

$$(\mathbf{q} \otimes \mathbf{p}) \otimes \mathbf{r} = \mathbf{p} \otimes (\mathbf{q} \otimes \mathbf{r}) \tag{A.9}$$

Definition A.4. Quaternion norm A quaternion norm $\|\mathbf{q}\|$ is

$$\|\mathbf{q}\| = \sqrt{q_x^2 + q_y^2 + q_z^2 + q_w^2} \tag{A.10}$$

Definition A.5. Unit quaternion A quaternion $\mathbf{q}$ is called a unit quaternion, denoted as $\overline{\mathbf{q}}$, if $\|\mathbf{q}\| = 1$

[1] Also *Euler-angles* can be used, which is a minimal representation of the rotation but they suffer from gimbal lock. Alternative is the rotation matrix which does not suffer from singularities like a gimbal lock but uses 9 numbers instead of minimal 3 numbers, causing problems when used for state estimation. Quaternions also do not suffer from singularities and use only 4 parameters, with small rotations parametrizable by 3 parameters.

An important fact is that only unit quaternions represent a proper rotation in 3D. Every rotation in 3D can be expressed as a rotation of $\boldsymbol{\theta}$ around vector $\mathbf{u}$. Corresponding quaternion is then

$$\overline{\mathbf{q}} = \begin{bmatrix} \mathbf{u} \sin \dfrac{\theta}{2} \\ \cos \dfrac{\theta}{2} \end{bmatrix} \tag{A.11}$$

The inverse rotation can be done by either rotating around $-\mathbf{u}$ or changing the rotation angle to $-\theta$. When both axis and rotation angle are inverted, a quaternion $-\mathbf{q}$ is produced, representing the same rotation as $\mathbf{q}$. That leads to another important fact that every rotation in 3D is represented by exactly two quaternions $\mathbf{q}$ and $-\mathbf{q}$.

Definition A.6. Rodrigues rotation formula A rotation matrix representing the same rotation as $\overline{\mathbf{q}}$ is

$$\mathbf{R} = \mathbf{I}_3 + \sin\theta \left\lfloor \mathbf{u}_\times \right\rfloor + (1 - \cos\theta) \left\lfloor \mathbf{u}_\times \right\rfloor^2 \tag{A.12}$$

A.2.1 Rotation matrix from the quaternion

A rotation matrix $\mathbf{R}$ corresponding to the quaternion $\mathbf{q}$ is often denoted as $\mathbf{R}\{\mathbf{q}\}$ or $\mathbf{C}(\mathbf{q})$.

Without a proof, we state that the following is true

$$\mathbf{R}\{1\} = \mathbf{I}_3 \tag{A.13}$$
$$\mathbf{R}\{-\mathbf{q}\} = \mathbf{R}\{\mathbf{q}\} \tag{A.14}$$
$$\mathbf{R}\{\mathbf{q}^\star\} = \mathbf{R}\{\mathbf{q}\}^T \tag{A.15}$$
$$\mathbf{R}\{\mathbf{q}_1 \otimes \mathbf{q}_2\} = \mathbf{R}\{\mathbf{q}_1\}\mathbf{R}\{\mathbf{q}_2\} \tag{A.16}$$

where $\cdot^\star$ denotes a conjugate of a quaternion.

A.2.2 Small rotations

When a given rotation is small ($\theta \to \infty$), both quaternion and its corresponding rotation matrix can be expanded with Taylor series, only keeping the constant and linear terms. With $\Delta\boldsymbol{\theta} = \theta\mathbf{u}$[2]

$$\Delta\mathbf{q} = \begin{bmatrix} \frac{1}{2}\Delta\boldsymbol{\theta} \\ 1 \end{bmatrix} + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|^2) \qquad \Delta\mathbf{R} = \mathbf{I}_3 - \left\lfloor \Delta\boldsymbol{\theta}_\times \right\rfloor + \mathcal{O}(\|\Delta\boldsymbol{\theta}\|^2) \tag{A.17}$$

A.2.3 Time derivatives

When the rotation is not constant over time, a quantity $\mathbf{q}(t)$ represents the rotation at a given time t. The angular rotation rate[3] $\boldsymbol{\omega}$ is a time derivative of

[2] Also called an angle-axis representation.

[3] The angular rotation rate is the quantity measured by the gyroscope.

$\mathbf{q}(t)$.

$$\boldsymbol{\omega}(t) = \lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{\theta}}{\Delta t} \tag{A.18}$$

A quaternion denoting the derivative of $\mathbf{q}(t)$ is $\dot{\mathbf{q}}(t)$, and $\dot{\mathbf{R}}(t)$ is the corresponding rotation matrix.

$$\dot{\mathbf{q}}(t) = \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}(t))\mathbf{q}(t) \qquad\qquad \dot{\mathbf{R}}(t) = \lfloor \boldsymbol{\omega}(t) \rfloor \, \mathbf{R}(t) \tag{A.19}$$

B. DVD content

The enclosed DVD contains more detailed results from all the experiments, including a source code of the application used for evaluation.

B.1 Experiment data

All the data related to the experiments described in this work are located in the directory `experiments`.

It contains the following items

- `experiments/datasets`: For each dataset, this directory contains two files. First one is the video file with the recording from the camera and the second one is a SQLite database [74] containing all the sensor outputs.
- `experiments/ground_truth_park_sequence.kmz`, `experiments/ground_truth_residential_sequence.kmz`:
  A KMZ files with the approximated ground truth for both sequences. It can be viewed for example using Google Earth Pro.
- `experiments/euroc_V2_02_medium_v2`, `experiments/euroc_V2_03_difficult_v2`:
  The results of the experiments on the EuRoC dataset from section 7.1.
- `experiments/park_no_gps`, `experiments/residential_no_gps`: The results of the park and residential experiments described in section 7.2, where the GPS was not used.
- `experiments/park_with_gps`, `experiments/residential_with_gps`: The results of the park and residential experiments described in section 7.3, where the GPS was used.
- `experiments/residential_with_gps_outage`: The GPS outage experiment from section 7.4, results on the residential sequence.
- `experiments/residential_with_gps_noise`: This folder contains 10 subfolders `trial_1`, ..., `trial_10` with the results for each trial of the experiment with high noise added to the measurements (section 7.5).

For each experiment, there are multiple files but not all experiments have all of them because of the differences in the input data.

- `viewer.avi`: A video output of the filter. The video is split into two parts. On the left hand side is the video stream from the camera with the information from the image processing. On the right hand side is the top-down view of the trajectory. This is not available for EuRoC dataset.
- `screen.avi`: A screencast of the 3D viewer. It contains the trajectory estimate and the triangulated positions of the detected and successfully processed features. This is not available for EuRoC dataset.

- `detailed_ground_truth_eval.html`: A HTML file with detailed comparison with the ground truth. It is only present for EuRoC dataset.
- `plot_state_time_series.html`: A HTML file with the time series of some of the variables estimated by the filter. It includes the uncertainty levels. The internet connection is needed to properly view this file.
- `trajectory.kml`: The output trajectory, aligned with the world. This file is not present for the EuRoc dataset. For the experiments without the GPS, the trajectory was aligned by hand. The GPS-based experiments contain the trajectory as estimated by the filter. It can be viewed for example using Google Earth Pro.
- `params.json`: A JSON file with parameters used to run the experiment.
- `raw_data.pkl`: The raw data from the filter, aligned with ground truth if available. The file is in the Python-specific serialization format Pickle version 3.

The video files are encoded using the H265 codec, which might not be available for some older video players.

B.2 Source code

The folder `code` contains the source code used to evaluate the experiments. For the details on how to use them, refer to the `code/README.md`.